ET0023 Operating Systems

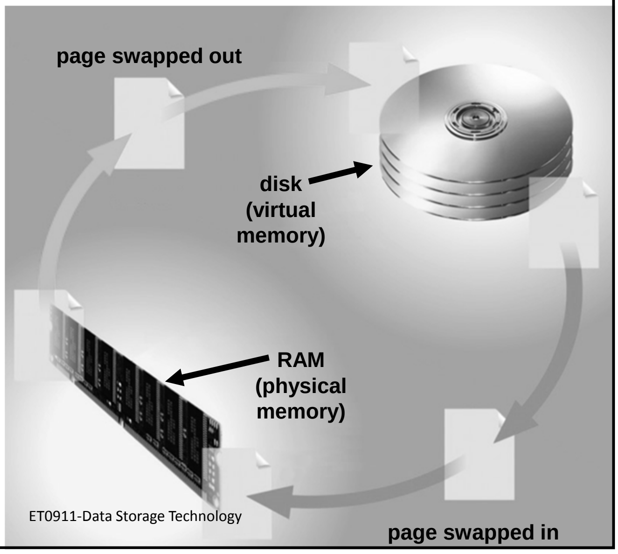
9. Virtual Memory

Virtual Memory

- Definition:
  - A storage allocation scheme in which secondary memory can be addressed as though it were part of main memory.
- Technique:
  - A program references memory using a virtual address.
  - The virtual address is translated automatically to the corresponding real memory address.
  - The size of virtual storage is limited by
    - the addressing scheme of the computer system, and
    - the amount of secondary memory available
    - not by the actual number of main storage locations.

What is virtual memory (VM) management?

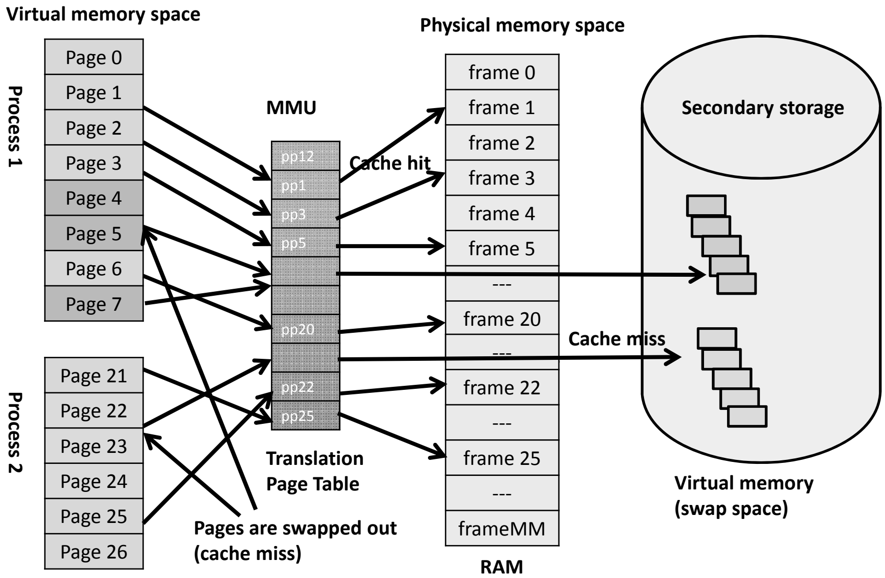
- Operating system allocates portion of hard disk to function like RAM
- Paging



Implementation

- Each process has an illusion of a large address space e.g. 2^{32} (for 32-bit addressing) = 4 GB
- But this illusion is also shared by multiple processes!
- However, physical address space is much smaller e.g. 4 GB
- So if we have 8 processes addressing 4GB each
  - total memory = 8 x 4 GB = 32 GB
  - But we have only 4 GB!!

Implementation



Advantages of Virtual Memory

- Separates user's logical memory from physical memory.
  - Only part of the program needs to be in physical memory for execution
  - Logical address space can therefore be much larger than physical address space
  - Allows physical address spaces to be shared by several concurrent processes
  - Allows for more efficient process creation

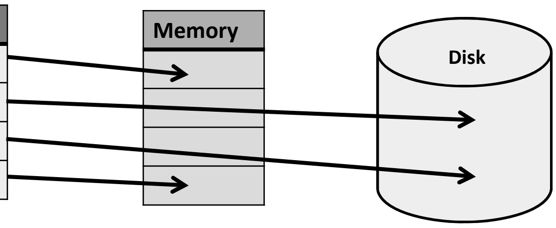
Swapping vs Paging

Swapping	Paging
• Loads entire process in memory: runs it, exit • Is slow (for big, long-lived processes) • Wasteful (might not require everything)	• Runs all processes concurrently, taking only pieces of memory (pages) away from each process • Finer granularity, higher performance • Paging completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory

How does VM work?

- Modify Page Tables with another bit ("is present")
  - If page in memory, is_present = 1, else is_present = 0
  - If page is in memory, translation works as before
  - If page is NOT in memory, translation causes a page fault

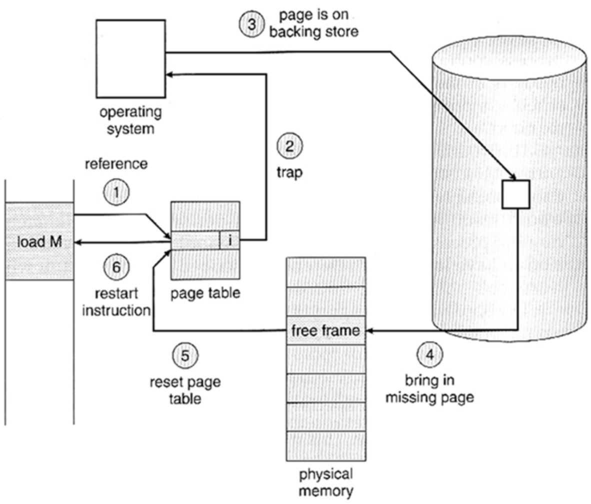
page	location	IP
Pg000	1273	1
Pg001	443	0
pg002	177	0
pg003	3000	1



How to handle Page Faults

- On a page fault:
  - OS finds a free frame, or evicts one from memory
    - Which one?
    - Want knowledge of the future?
  - Issues disk request to fetch data for page
    - What to fetch?
    - Just the requested page, or more?
  - Block current process, context switch to new process
    - How?
    - Process might be executing an instruction
  - When disk completes, set is_present bit to 1, and current process in ready queue

Steps in handling a Page Fault



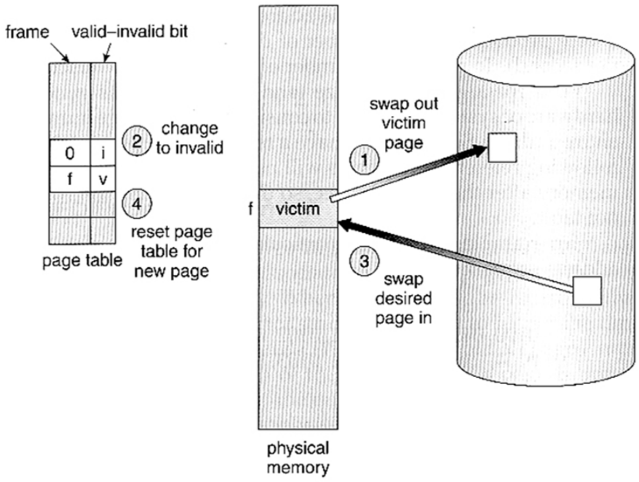
What to replace (who to evict)?

- What happens if there is no free frame?
  - Find a suitable page in memory, swap it out
- Page Replacement
  - When process has used up all frames it is allowed to use
  - OS must select a page to eject from memory to allow new page
  - The page to eject is selected using the Page Replacement Algorithm
- Goal: Select page that minimizes future page faults

Tip: Modify/Dirty Bits

- Evict pages that have not been changed!
- Use modify (dirty) bit to reduce overhead of page transfers
- Only modified pages are written to disk. Writing has high overheads.
- Process text segments are rarely modified, can bring pages back from the program image stored on disk

Page Replacement

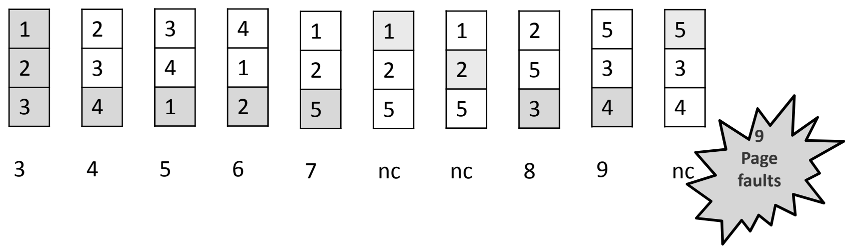


Page Replacement Algorithms

• Random	Pick any page to eject. Used for comparison.
• FIFO	The page brought in earliest is evicted Ignores usage.
• OPT	Belady's algorithm Select page not used for longest time
• LRU	Evict page that hasn't been used the longest. Past could be a good predictor of the future
• MRU	Evict the most recently used page
• LFU	Evict least frequently used page

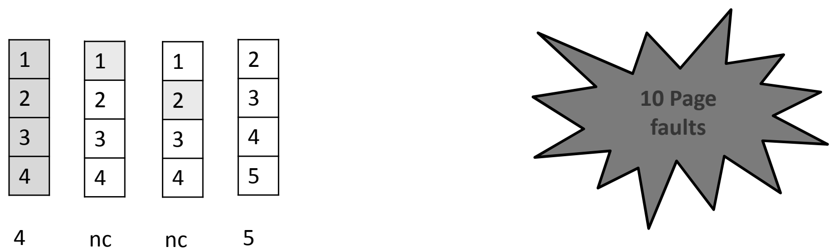
First-In-First-Out Replacement

- Assume pages are accessed in the following order: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- We only have 3 frames in memory per process
- How many page faults?



First-In-First-Out Replacement

- Assume pages are accessed in the following order: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- We only have 4 frames in memory per process
- How many page faults?



Optimal Algorithm

- Replace page that will not be used for longest period of time
- How do we know which frame?
- Used for measuring how well your algorithm performs
- Eg: 4 frames, Seq: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Result: 6 Page faults

Optimal Approximation

- In real life, we do not have access to the future page request stream of a program
- No way to know definitively which pages a program will access
- So we need to make a **best guess** at which pages will not be used for the longest time

Least Recently Used Algorithm

- Implemented with a Counter.
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
  - When a page needs to be changed, look at the counters to determine which are to change
- Eg: 4 frames, Seq: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	2	3	4	4	4	5	1	2
2	3	4	1	2	5	1	2	3
3	4	1	2	5	1	2	3	4
4	1	2	5	1	2	3	4	5
4	nc	nc	5	nc	nc	6	7	8

8 Page faults

Implementing Perfect LRU

- On reference – timestamp the page
- On eviction – scan for oldest frame
- Problems:
  - Large page lists (scanning is slow)
  - Timestamps are costly (software/implementation)
- Possible solutions
  - Clock Algorithm (FIFO + reference bit), low accuracy
  - Clock Algorithm with another hand – Leading edge clears reference bits, trailing edge evicts pages

Other Algorithms

- MRU (Most Recently Used)
  - Works well for data that is accessed only once
  - Not good fit for other data
- LFU (Lowest Frequently Used)
  - Remove pages with lowest count
  - No timestamp, just a counter (shift right by 1)
- MFU (Most Frequently Used)
  - Not a good algorithm

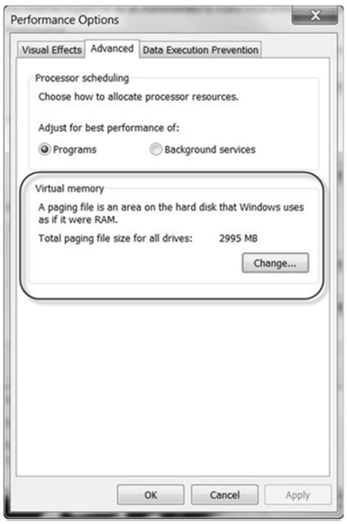
Allocating Pages to Processes

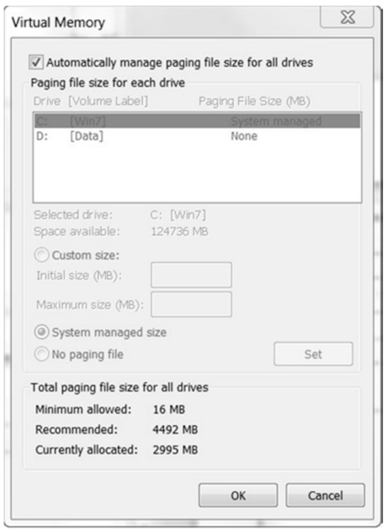
- Global replacement
  - Single memory pool for the entire system
  - On page fault, evict oldest page in system
  - Problem: Lack of performance isolation
- Local (per-process) replacement
  - Have separate pool of pages for each process
  - Page fault in one process, can only replace pages from its own process
  - Problem: May have idle resources

Thrashing

- Definition: Excessive rate of paging
  - Lack of resources
  - Caused by bad choices of the eviction algorithm
  - Keep throwing out page that will be referenced soon
  - Swap-in and Swap-out of pages
- Causes
  - Poor locality, present requirements does not suit future processing
  - There is reuse, but process does not fit model
  - Too many processes in the system

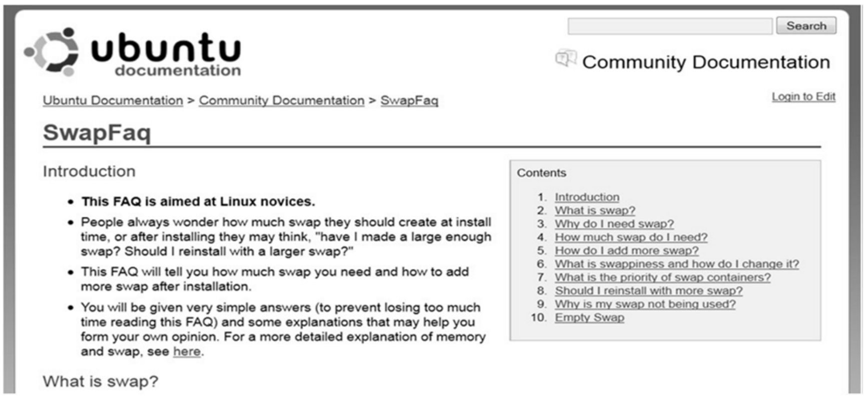
Windows





Linux

- Swap Space
- Read: https://help.ubuntu.com/community/SwapFaq



QUESTIONS