Virtual Memory

1- Logical address can be dynamically translated into physical addresses using various hardware and software techniques.

2- A process can be broken up into multiple pieces and they need not be in contiguously located in the main memory.

If these two characteristics are met it is not necessary to have all the pieces (pages, segments, etc.) of the program to be in the memory during execution.

- The idea! when we want to run a program, OS brings a piece of process in to the Main memory (MM)
- This is called Resident Set of the process.
- As long as all the memory accesses are inside the resident set process executes smoothly.
- If the CPU encounters an address that is not in the resident set, it generates an interrupt indicating memory access fault.
- OS puts the interrupted process into blocking state.
- OS brings the piece that was requested from the main memory. (I/O request)
- while this I/O request is being processed OS can dispatch another process, to run while the disk I/O is performed.

- Once I/O request is finished, I/O interrupt is issued, giving control back to the ~~OS~~ OS, which places the affected process back to ready state.
  - More processes in main memory
  - A process can be larger than the MM.

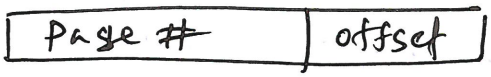
This strategy effectively uses ~~bothe~~ both main memory and secondary storage.

However, from the process point of view, it views the memory as contiguous block of memory.

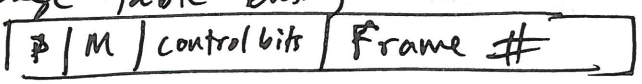
This is called virtual memory.

Thrashing
- The system spends more time swapping ~~pieces~~ pieces of memory rather than executing the process.

- practically, OS designers use paging to implement virtual memory.
- In simple paging, process must be fully in the MM before it executes, unless programmer has used overlaying
- Hence, some extra information about pages must be kept in page tables to ensure VM idea working.



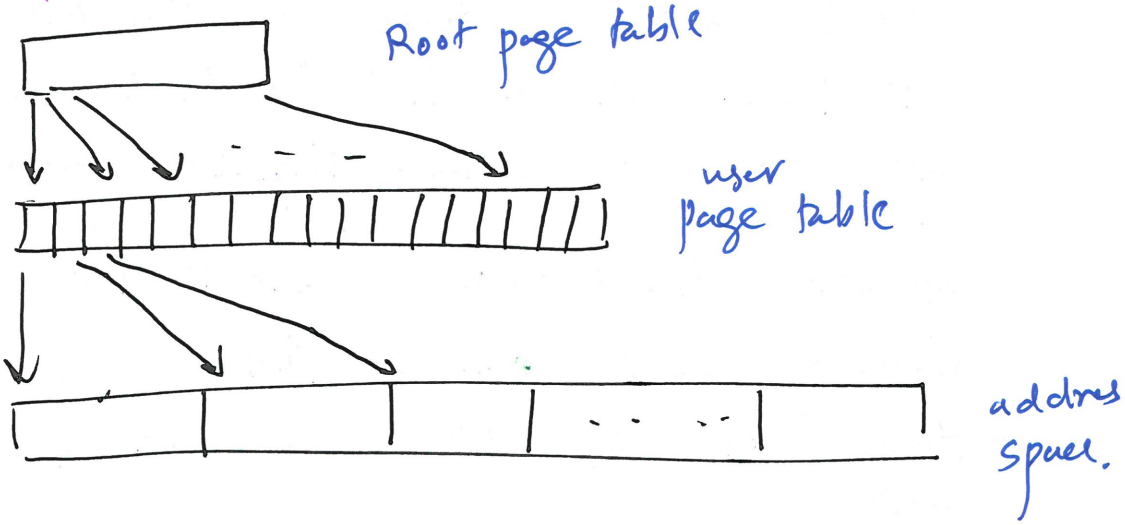
Page table entry



P - present bit
M - modified bits

- Page table is of variable length (depends on the size of the process)
- Mainly it is kept in MM.
- The page table pointer keeps the starting address of the page table.
- If there are lot of processes with large page tables, we have to keep enough space in the MM. The amount of memory that is dedicated to page tables is too high.
- Hence, most systems keep page tables in Virtual memory as well.
- Basically page tables can be subjected to paging as well.

- Two-level Hierarchical page table
  - Root page table has to be in the main memory.



Inverted page tables

- Alternative approach one- or multiple-level page tables.
- Idea, is there is one entry for each frame of the MM.
- Each entry keeps track of which frame is assigned to which page.
- page # is mapped to an entry by a Hash-fun on
- If multiple pages are matched to same entry, chaining is used.

Translation Lookaside Buffer

- The idea is to speed up memory lookup.
- Even with page tables, and the best case scenario, every memory access to the MM, requires at least 2 access(es) to MM. (one for page table and one for actual physical memory location)
- specific hardware support for this using a memory cache called TLB.
- Each slot of this cache can keep a page table entory.
- Now, CPU first looks in TLB to check page entry exists before looking in page table.
- TLB is faster than MM access.

- TLB uses associative mapping.
- Basically TLB has special circuitary that allows CPU to search all the slots parallely. (very fast)

Things we have to consider when developing memory management software for OS.

Fetch policy - ~~How~~ How many/what pages should we bring to MM

- Demand paging: Bring only the page that is requested
  - Initially lot of page faults, then less
- prepaging: Bring pages other than the requested one, hoping process will uses additional pages that we bring in future.
  - Uses principle of locality
  - If the pages of process are stored contiguously in secondary storage (SS), this strategy is very useful.
  - will be ineffective, if the additional pages that we bring are not referenced.

Placement policy - where do we put the ~~process~~ pieces of process that we bring
- Not important in paging or paging + segmentation system
- In pure segmentation system this matters.

Replacement policy: ~~How~~ Which piece of memory are we going to swap when new piece needs to be brought in (specially, if they do not have enough space)

Basic Algos
- OPT (Needs to know about future memory references)
- LRU (~~Needs~~ Lot of overhead)
- FIFO (Easy to implement, but inefficient)
- CLOCK (better than FIFO)

Resident Set management

Resident set size
- fixed - fixed # of frames are allocated to proc.
- variable - variable # " " " " "

Replacement Scope
- Global (for fixed resident set size, global replacement policy is not used)
- Local

cleaning policy: Determine when modified page (piece of memory) should be written to SS.
- Demand cleaning
- precleaning

(similar to fetch policy)