*National University of Singapore*

# CS2106 Operating System

Second Half Summary Notes

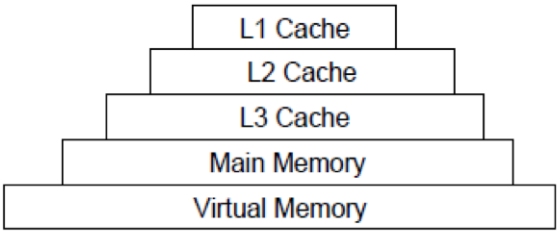
*Dong Shaocong* A0148008J

April 22, 2018

## 1 Virtual Memory

**Definition 1.1. Memory Hierarchy**

- Topmost layer is the fastest, but most expensive and therefore the smallest.
- Bottom-most layer is the cheapest and biggest, but the slowest.
- Each layer above contains a small portion of the layer below: Use **"replacement policies"** to decide which portions to copy.




**Definition 1.2. Principles of Locality**

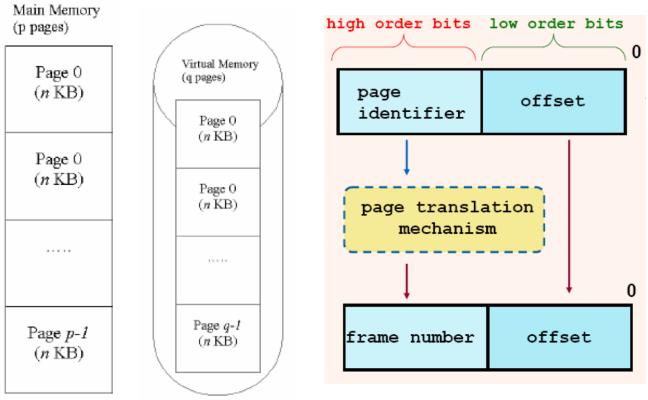
- **Spatial Locality**: If you've accessed a memory location, there's a very high chance that the next location you access is right next to it. E.g. executing instructions sequentially, accessing elements of an array.
- **Temporal Locality**: If you've accessed a memory location, there's a very high chance that you will access it again. E.g. Loops.
- **Note**: Because of locality, memory hierarchy allows you to have:
  - **Very fast memory**, since most accesses come from the fast top layer.
  - **Very large memory**, since we can continue to keep what we don't (yet) need in the lowest layer.

**Definition 1.3. Virtual Memory** It is implemented on your hard disk! The layer above (your "main memory") maintains a copy of a small portion of the VM.

We can do this by having instructions and data that the CPU is currently interested in, in main memory. Everything else stays on the VM. In this way we can squeeze MUCH MORE instructions and data than otherwise possible!

**Paging**

- VM is divided into fixed equal sized blocks called "pages".
- Physical memory is divided into "frames" that are the same size as a VM page. A "Virtual page" in the VM can be loaded to any "physical frame" in main memory.
- The CPU always generates "virtual addresses". I.e. any CPU address always points to a page in virtual memory.

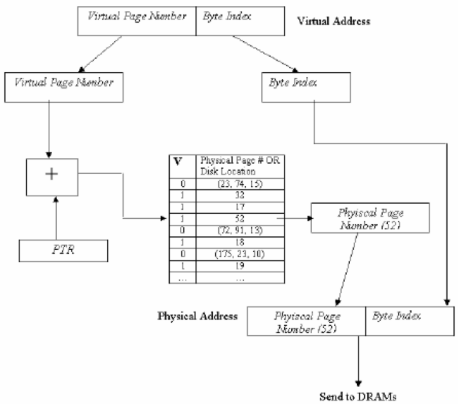


**Virtual to Physical Address Mapping**

| V | Physical Page # OR Disk Location |
|---|---|
| 0 | (23, 74, 15) |
| 1 | 32 |
| 1 | 17 |
| 1 | 52 |
| 0 | (72, 91, 13) |
| 1 | 18 |
| 0 | (175, 23, 10) |
| 1 | 19 |
| ... | ... |

- Virtual Page Number = Page Identifier; Frame number = Physical Page Number; Offset = Byte Index.
- Specialised hardware on the CPU, together with virtual memory services in the OS, work together to translate virtual addresses into physical addresses that correspond to locations in main memory.
- "page table": V=1 means the information requested is in main memory.

1. The virtual address forms an index into the page table. If the "virtual page" is in memory, a "memory translation" process takes place that locates which frame this virtual page has been loaded into.
2. This information is used to generate the "physical address" to access main memory.




- Given an N bit virtual addressing space, P bit physical addressing space with B byte page/frame size:

$$\#\text{ of bits in offset} = \log_2(B)$$

$$\#\text{ of bits in page identifier (or VPN)} = N - \log_2(B)$$

$$\#\text{ of bits in frame number} = P - \log_2(B)$$

- So if we have 32KB pages, a 4GB virtual addressing space and 2 GB of physical memory:

$$\text{offset} = log2(32KB) = 15 \text{ bits}$$

$$\text{Page identifier} : \log_2(4GB) - 15 = 32 - 15 = 17 \text{ bits}$$

$$\text{Frame number} : \log_2(2GB) - 15 = 31 - 15 = 16 \text{ bits}$$

  There will be $2^{17}$ pages in this system, with $2^{16}$ frames.

- **Note**:
  1. **No external fragmentation**: Pages that should be contiguous can be mapped to non-contiguous frames.
  2. **Internal fragmentation**: Basic allocation unit is now 1 page. Can be quite large!
  3. **Mapping is transparent** to programs. Programs only "see" virtual addresses.
  4. Can grow process segments by adding more pages. Growth is now limited to multiples of one page.
- **Page Faults**: The V flag will be "0". The hardware sees this and generates a "page fault" interrupt. This is vectored to a page fault ISR within the OS.
  1. Locates where the missing page is on the disk: When V=0, the page table contains the location on disk where the page is (in cylinder/side/block format).
  2. Finds a free frame to load the VM page into.
  3. Updates the page table. Set V=1, and changes the entry to show which frame the virtual page has been loaded into.

4. The VM then goes through the rest of the address translation process to allow the CPU to access the faulting data/instruction.

- **Page loading policy**:
  - **Demand Paging**: Page is loaded when an access is made to a location inside it.
  - **Pre-paging**: Other pages (e.g. surrounding pages) can be loaded together with the fault page. Pages can be pre-loaded when a process starts.
  - Can be a mix of strategies.
- **Replacement policy**:
  - **FIFO**:
    - **Principle**: First page in = first page out.
    - Use a FIFO queue of pages read in. New pages are added to the tail, pages at the head are swapped out when no more frames.
    - **Belady's anomaly**: In a FIFO replacement policy, having more frames can increase paging instead of decreasing it.
    - **Resolution**: consider frequency of use (i.e. temporal locality) instead of age. Optimal Page Replacement (OPT), Least Recently Used (LRU) and Clock Replacement (CR). None of these suffer from Belady's Anomaly.
  - **LRU - Least Recently Used**
    - Each page table entry (PTE) has an p-bit counter $c_i$. At each access to page i, the $c_i$ is set to $2^p - 1$. Counter for all other pages is decremented by 1.
    - When it is time to replace a page: Perform a search through page table to find smallest $c_i$. Swap that page back to disk.
    - If $> 1$ page has smallest $c_i$, choose the first one we encountered.
    - **Note**: only approximation version of LRU as the counting range (p bit) cannot be too big.
- **Rewriting pages back to disk**: Pages are large. Writing swapped out pages back to disk is expensive. Have a "D" (Dirty) bit in each PTE. **Note**: requires hardware support to update D.

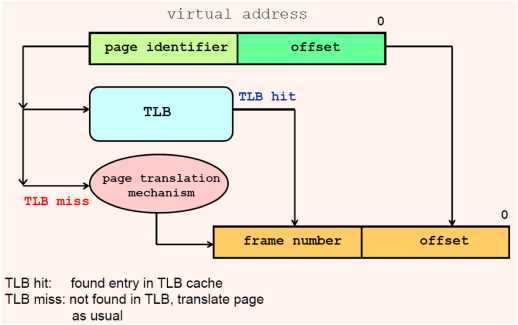
**Definition 1.4. Thrashing**: a performance disaster When the amount of data/instructions in VM is much, much greater than available physical memory.

1. Accessing instructions/data might frequently be in pages that aren't yet in physical memory.
2. Shortage of physical memory causes frames to be frequently swapped out to disk.
3. The swapped out frames are accessed again, causing other frames to be swapped out so that these can be swapped back in.
4. Huge array, 4 million bytes. Random accesses will likely cause pages to be swapped out and back in again.

**Definition 1.5. Translation Lookaside**

The page table is in main memory. **Two access**: One access to consult the page table. One access to actually read/write the physical memory. (Main memory is slower than the cache, therefore store parts of the page table in cache)

- This special cache is called the "Translation Lookaside Buffer" or TLB. This is often located on the CPU die itself.



**Example 1.1. Virtual memory calculation example**

- **Question**: consider a virtual memory system with 64 bytes per page, 10 bit virtual addresses and 9 bit physical addresses.
- Maximum size in bytes of the virtual memory is $2^{10} = 1024$ bytes. Maximum size of the physical memory is $2^9 = 512$ bytes.
- Byte index is 6 bits ($2^6 = 64$), leaving 4 bits for VPN or 16 virtual pages, and 3 bits for PPN or 8 physical pages.
- **translation procedure**: convert virtual address into binary → get the VPN → look up at the page table to get the PPN → add the byte index (last few bits just now) → convert the PPN and byte index from binary to decimal, this is the physical address.