Paging: TLB and Multi-Level Page Tables; Swapping

Refresher Concepts:

VA to PA translation:

1. Extract VPN and offset from the virtual address.
2. Use the VPN to index into the page table (using the PTBR).
3. Obtain the PFN from the page table entry (PTE).
4. Concatenate PFN + offset to form the physical address.

As shown in midsem rubric, you can also use the formula to calculate the PA (PFN*Page Size+Offset).

Translation Lookaside Buffer (TLB)

- TLB = hardware cache for recent virtual-to-physical translations.
- Checked first on each memory access.
- TLB **hit** -> translation done quickly (no page-table access).
- TLB **miss** -> page-table lookup is needed.
- TLBs are fully associative, typically with 32, 64, or 128 entries

Locality Principles

- Spatial locality: addresses near recently used ones are likely to be accessed soon.
- Temporal locality: recently accessed items are likely to be reused soon.

Handling Misses

- Hardware-managed TLBs: handled entirely by hardware using the Page Table Base Register (PTBR).
- Software-managed TLBs: hardware raises a trap; the OS finds the PTE, updates the TLB, and resumes execution.
- Handlers are kept in physical memory or pinned in the TLB to avoid recursive/infinite misses.

Context Switches

- TLB contains VA to PA translations that are only valid for the current running process.
- *What about other processes? How can this be solved?*
- Solution 1: Flush TLB on context switch. HOWEVER, this causes a performance overload, and by flushing the TLB on every context switch, many translations become invalid for the new processes, thus leading to even more misses. Now, how can *this* be solved?

- Solution 2: Share the TLB across context switches by adding an ASID (Address-Space Identifier) field in each TLB entry to tag specific processes! ✅

TLB Replacement

- LRU policy: evict the least-recently-used translation.
- Random policy: evict any entry randomly.

Page Table Size Example (32-bit system)

Given:

- Virtual address = 32 bits
- Page size = 4 KB (2^{12})
- Page table entry (PTE) = 4 bytes
- Size of data stored in each address location = 1 byte

How to calculate:

- Number of unique addresses possible = 2^32 (32 bits, 0 and 1, two possibilities for each bit).
- Virtual Memory Size = number of unique addresses possible x size of data stored in each location = 2^32 x 1 = 2^32
- Number of virtual pages = VA space/page size = 2^32/2^12 = 2^{20}.
- Page-table size = Number of pages x size of each PTE = $2^{20} \times 4$ = 4 MB.
- VA bits = log_base2(Virtual Memory Size in bytes) = log(2^32) = 32
- Offset bits = log_base2(page size in bytes) = log(2^12) = 12.
- Number of bits in a virtual address = VPN bits = VA bits - Offset bits= 32-12 = 20

Multi-Level Page Tables

- Split the page table into page-sized chunks.
- Allocate only portions that are needed.
- Page directory points to valid page-table pages.
- *Advantages: saves memory; page tables need not be contiguous.*
- *Disadvantages: more memory accesses per translation (one for PDE + one for PTE).*
- *Trade-off: time vs space.*

Multi-level Page Table Example:

Given:

- VA Space = 16 KB
- Page size = 64 bytes
- PTE: 4 bytes

How to calculate:

- VA bits = 14 (16 KB = 2^{14}).
- Offset bits = log_base2(page size in bytes) = log(2^6) = 6
- VPN bits = 8 (i.e., 14 - 6)
- Number of pages = VA space/page size = 16KB/64B = 2^14/2^6 = 2^8
- Then, linear page table size = number of pages x size of PTE = $2^8 \times 4$ = 2^10 = 1024 = 1 KB.
- Number of pages in the linear page table = size of linear page table / page size = 2^10/2^6 = 2^4
- 64-byte pages hold 64/4 = 16 PTEs each
- And then the page directory will have 2^4 entries, which is also equal to the number of pages in the page table.

Inverted Page Tables

- A single-page table that has an entry for each physical page of the system.
- Each entry stores:
    - Which process owns the frame (PID).
    - Which virtual page maps to it.
- Lookup uses a linear or hash search.
- Advantage: table size is proportional to the number of physical frames (saves memory).
- Disadvantage: slower lookups.

Swapping:

- So far, we assumed all process pages fit in physical memory. To support larger virtual memory, the OS moves pages between RAM and disk (swap space).

Swap Space:

- Reserved disk area used to store pages temporarily evicted from RAM.
- OS reads/writes pages in page-sized units.
- The PTE stores the disk address of the swapped-out page.

Present Bit:

- Added to each PTE.
- 1 = page in memory, 0 = page on disk.
- On a page fault, the OS:

    1. Finds the disk address from PTE.
    2. Reads the page into memory.
    3. Updates PTE (set Present = 1, update PFN).

Page Fault:

- Occurs when a referenced page is not present in memory.
- Serviced by the page-fault handler in the OS.
- If memory is full, the OS must evict an existing page.

Page Replacement Policies:

1. Optimal (minimizer): replace the page that will not be used for the longest time in the future, theoretical best.
2. FIFO: evict the oldest page in memory (first in, first out).
3. LRU: evict the least-recently-used page (uses past behavior as a predictor).
4. LFU: evict the least-frequently-used page.

Implementing LRU:

- Option 1: maintain a linked list (move most-recently-used page to the front).
- Option 2: record a timestamp for each page.
- Both methods are expensive in hardware/software.

Approximating LRU, Clock Algorithm:

- Each page has a use/reference bit in its PTE.
- Pages are arranged in a circular list.
- The OS scans the list and evicts the first page with the use bit = 0.
- When it encounters a page with use bit = 1, it resets it to 0 and continues.
- Dirty bit marks modified pages; the algorithm prefers clean (non-dirty) pages for eviction.

Other Terms:

- Demand paging: bring a page into memory only when needed.
- Thrashing: occurs when the working set of all processes exceeds physical memory, causing continuous paging activity.

Questions:

Q1. A system has a 16KB virtual address space, with page size = 64 bytes. Each page table entry (PTE) is 4 bytes.

1. How many bits are required to represent a virtual address?
2. How many virtual pages are there?
3. What is the size of a linear page table (in bytes)?
4. If multi-level paging is used with page size = 64 bytes, how many entries can fit in a single page table page?

Ans.

1. Number of bits in VA

    - Virtual address space = 16KB = 2^14 bytes.
    - 14 bits required.
2. Number of virtual pages
    - Page size = 64 bytes = 2^6
    - Number of virtual pages = VA space/page size = 2^14/2^6=2^8=256
    - 256 virtual pages.
3. Size of linear page table
    - One PTE = 4 bytes.
    - For 256 virtual pages: 256×4=1024 bytes = 2^10.
    - 1024 bytes (1 KB).
4. Number of PTEs per page
    - Page size = 64 bytes.
    - Each PTE = 4 bytes.
    - Entries per page = 64/4=16.
    - 16 entries per page table page.

Q2. Consider a system with 16-bit virtual addresses, 256-byte pages, and 4-byte page table entries. The OS builds a multi-level page table for each process. Calculate the maximum number of pages required to store all levels of the page table of a process in this system.

Ans: Number of PTE per process = 2^16/2^8 = 2^8. Number of PTE per page = 2^8/2^2 = 2^6. Number of inner page table pages = 2^8/2^6 = 4, which requires one outer page directory. So total pages = 4+1 = 5.

Q3. Consider a process with 4 logical pages, numbered 0–3. The page table of the process consists of the following logical page number to physical frame number mappings: (0, 11), (1, 35), (2, 3), (3, 1). The process runs on a system with 16-bit virtual addresses and a page size of 256 bytes. You are given that this process accesses virtual address 770. Answer the following questions, showing suitable calculations.
(a) Which logical page number does this virtual address correspond to?
(b) Which physical address does this virtual address translate to?

Ans:
(a) 770 = 512 + 256 + 2 = 00000011 00000010 = page 3, offset 2
(b) page 3 maps to frame 1. physical address = 0000001 00000010 = 256 + 2 = 258

Q4. The page size in a system (running a Linux-like operating system on x86 hardware) is increased while keeping everything else (including the total size of main memory) the same. For each of the following metrics below, indicate whether the metric is generally expected to increase, decrease, or not change as a result of this increase in page size.
(a) Size of the page table of a process
(b) TLB hit rate
(c) Internal fragmentation of main memory

Ans:
(a) PT size decreases (fewer entries)
(b) TLB hit rate increases (more coverage)
(c) Internal fragmentation increases (more space wasted in a page)

Q5. Consider a system with a 6-bit virtual address space and 16-byte pages/frames. The mapping from virtual page numbers to physical frame numbers of a process is (0,8), (1,3), (2,11), and (3,1). Translate the following virtual addresses to physical addresses. Note that all addresses are in decimal. You may write your answer in decimal or binary.
(a) 20 (b) 40

Ans:
(a) 20 = 01 0100 = 11 0100 = 52
(b) 40 = 10 1000 = 1011 1000 = 184

Q6. Consider a system where each process has a virtual address space of 2 v bytes. The physical address space of the system is 2 p bytes, and the page size is 2 k bytes. The size of each page table entry is 2 e bytes. The system uses hierarchical paging with l levels of page tables, where the page table entries in the last level point to the actual physical pages of the process. Assume l ≥ 2. Let v0 denote the number of (most significant) bits of the virtual address that are used as an index into the outermost page table during address translation. (a) What is the number of logical pages of a process? (b) What is the number of physical frames in the system? (c) What is the number of PTEs that can be stored in a page? (d) How many pages are required to store the innermost PTEs? (e) Derive an expression for l in terms of v, p, k, and e. (f) Derive an expression for v0 in terms of l, v, p, k, and e.

Ans: (a) 2^(v−k)
(b) 2^(p−k)
(c) 2(k−e)
(d) 2^(v−k) / 2^(k−e) = 2^(v+e−2k)
(e) The least significant k of v bits indicates an offset within a page. Of the remaining v-k bits, k-e bits will be used to index into the page tables at every level, so the number of levels l = *ceil* (v−k)/(k−e)
(f) $v - k - (l - 1) * (k - e)$

Q7. Consider an operating system that uses 48-bit virtual addresses and 16KB pages. The system uses a hierarchical page table design to store all the page table entries of a process, and each page table entry is 4 bytes in size. What is the total number of pages that are required to store the page table entries of a process across all levels of the hierarchical page table?
Ans: $2^{22} + 2^{10} + 1$

Virtual Address space is 48-bit and page size = 16 KB = 2^{14} bytes. So, the total number of page table entries = $2^{48}/2^{14} = 2^{34}$.

Each page table entry is 4 bytes, and the page size is 16 KB, so the number of entries stored in one page table = 16KB/4 = 2^{12} page table entries.

Lowest level: So, the number of pages at the lowest level of the hierarchical table would contain entries for all 2^{34} virtual pages, so the number of pages at the lowest level = $2^{34}/2^{12} = 2^{22}$.

Second-lowest level: Now, pointers to all these lowest-level pages must be stored in the next level of the page table, so the next level of the page table has $2^{22}/2^{12} = 2^{10}$ pages.

Third-lowerst level: Finally, these pointers must be stored in the next level of the page table, so this would have $2^{10}/2^{12} < 1$. Therefore, a single page can store all the 2^{10} page table entries, so the outermost level has one page.

So, the total number of pages that store page table entries is $2^{22} + 2^{10} + 1$.

Q8. Using the least frequently used (LFU) policy for page replacement, find the number of hits, misses, and the hit rate for the following pages. The cache/frame size is 3. Show all the steps, including the state of the cache at each step, along with indicating the frequency of the page. Clearly state your assumptions, for example, in the case of a tie.

Page sequence: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1

Ans. Assumptions: In case of a tie, we apply LRU here and choose the page that was least recently used/accessed. Other possibilities include applying FIFO or picking one at random. The answer can change depending on the tie-breaking mechanism used.

Step	Page	Cache State (after insertion)	Frequency (after insertion)	Hit/Miss
1	7	[7]	{7: 1}	Miss
2	0	[7, 0]	{7: 1, 0: 1}	Miss
3	1	[7, 0, 1]	{7: 1, 0: 1, 1: 1}	Miss
4	2	[0, 1, 2]	{0: 1, 1: 1, 2: 1}	Miss (Replace 7)
5	0	[0, 1, 2]	{0: 2, 1: 1, 2: 1}	Hit
6	3	[0, 2, 3]	{0: 2, 2: 1, 3: 1}	Miss (Replace 1)
7	0	[0, 2, 3]	{0: 3, 2: 1, 3: 1}	Hit
8	4	[0, 3, 4]	{0: 3, 3: 1, 4: 1}	Miss (Replace 2)

9	2	[0, 4, 2]	{0: 3, 2: 2, 4: 1}	Miss (Replace 3)
10	3	[0, 2, 3]	{0: 3, 2: 2, 3: 2}	Miss (Replace 4)
11	0	[0, 2, 3]	{0: 4, 2: 2, 3: 2}	Hit
12	3	[0, 3, 2]	{0: 4, 3: 3, 2: 2}	Hit
13	2	[0, 3, 2]	{0: 4, 3: 3, 2: 3}	Hit
14	1	[0, 2, 1]	{0: 4, 2: 3, 1: 2}	Miss (Replace 3)

Hits: 5 (pages: 0, 0, 0, 3, 2)

- Misses: 9 (pages: 7, 0, 1, 2, 3, 4, 2, 3, 1)
- Total requests: 14
- Hit Rate = (Number of Hits) / (Total requests) = 5/14 = 0.3571
- Final frequency : {7:1, 0: 4, 1:2, 2:3, 3:3, 4:1}

Q9. Use the clock algorithm that considers reference/use and dirty bits for replacing pages. Find the number of hits, misses and the hit rate for the following pages. The cache/frame size is 4. The reference bit of all the pages is initially set to 1. The dirty bit of pages 1, 2, 3 are set to 1 and is 0 for the rest of the pages. Show all the steps, including the state of the cache at each step, the reference bit and use bit for each page. Clearly state your assumptions, for example, the position of the hand.

Page sequence: 1, 2, 3, 4, 5, 1, 6, 2, 7, 8, 1, 2, 9, 10

Ans. Assumptions: Clock hand always starts with page 1 in the cache.

Step	Page	Cache state after insertion - (page, use bit, dirty bit)	Hit/Miss	Comments
1	1	(1,1,1)	Miss	
2	2	((1,1,1), (2,1,1))	Miss	
3	3	((1,1,1), (2,1,1), (3,1,1))	Miss	
4	4	((1,1,1), (2,1,1), (3,1,1), (4,1,0))	Miss	
5	5	((1,0,1), (2,0,1), (3,0,1), (5,1,0))	Miss	Replace 4 in the second pass since that is the only page with dirty bit 0.
6	1	((1,1,1), (2,0,1), (3,0,1), (5,1,0))	Hit	
7	6	((1,0,1), (2,0,1), (3,0,1), (6,1,0))	Miss	Replace 5 in the second pass since that is the only page with dirty bit 0.
8	2	((1,0,1), (2,1,1), (3,0,1), (6,1,0))	Hit	
9	7	((1,0,1), (2,0,1), (3,0,1), (7,1,0))	Miss	Replace 6 in the second pass since that is the only page with dirty bit 0.
10	8	((1,0,1), (2,0,1), (3,0,1), (8,1,0))	Miss	Replace 7 in the second pass since that is the only page with dirty bit 0.
11	1	((1,1,1), (2,0,1), (3,0,1), (8,1,0))	Hit	
12	2	((1,1,1), (2,1,1), (3,0,1), (8,1,0))	Hit	
13	9	((1,0,1), (2,0,1), (3,0,1), (9,1,0))	Miss	Replace 8 in the second pass since that is the only page with dirty bit 0.
14	10	((1,0,1), (2,0,1), (3,0,1), (10,1,0))	Miss	Replace 9 in the second pass since that is the only page with dirty bit 0.

- Hits: 4
- Total requests: 14
- Hit Rate = (Number of Hits) / (Total requests) = 0.286