CSEN177, Winter 2025 Homework 5

1. With a 1-KB page size, each page holds 1,024 bytes. To find the page number and offset for a given decimal address AAA:

   Page number = A/1024

   Offset = A mod 1024

A) A = 3085

   Page number = (3085 / 1024) = 3

   Offset = (3085 mod 1024) = 13

B) A = 42095

   Page number = (42095 / 1024) = 41

   Offset = (42095 mod 1024) = 111

C) A = 215201

   Page number = (215201 / 1024) = 210

   Offset = (215201 mod 1024) = 161

D) A = 650000

   Page number = (650000 / 1024) = 634

   Offset = (650000 mod 1024) = 784

E) A = 2000001

   Page number = (2000001 / 1024) = 1953

   Offset = (2000001 mod 1024) = 129

2. a) Since there are 256 pages, we need 8 bits to identify a page (because 256 = 2^8).
   4-KB page size means each page holds 4 × 1024 = 4096 bytes. Since 4096 is 2 raised to the power of 12, we will need 12 bits to specify the offset within a 4-KB page.
   Therefore, total bits in the logical address = page bits + offset bits = 8 + 12 = 20 bits.

b) The system has 64 frames, which is 2^6. Thus, 6 bits are needed for the frame number. We again need 12 bits for the offset. Therefore, total bits in the physical address = 6 + 12 = 18 bits.

3. Internal fragmentation occurs when a process is allocated a block of memory that is larger than it actually needs, this results in unused or wasted space inside that allocated block. External fragmentation happens when free memory becomes broken up into small noncontiguous chunks over time, making it difficult to find a sufficiently large contiguous block for new allocations, even if the total amount of free memory is large enough.
4. FIRST-FIT: We scan from the left, picking the first partition that is big enough:

Process 1 (200 MB):

- Skip partitions 1 (100 MB), 2 (170 MB), 3 (40 MB).
- Partition 4 (205 MB) is big enough → allocate 200 MB there.

Process 2 (15 MB):

- Partition 1 (100 MB) is the first big enough → allocate 15 MB there.

Process 3 (185 MB):

- Skip leftover in partition 1 (now 85 MB), skip partition 2 (170 MB), skip partition 3 (40 MB).
- Partition 5 (300 MB) is big enough → allocate 185 MB there.

Process 4 (75 MB):

- Go back to leftover in partition 1 (85 MB), which is big enough → allocate 75 MB there.

Process 5 (175 MB):

- Skip leftover 10 MB in partition 1, skip partition 2 (170 MB, too small), skip partition 3 (40 MB).
- Skip leftover 5 MB in partition 4. Skip leftover 115 MB in partition 5 (too small).
- Partition 6 (185 MB) is big enough → allocate 175 MB there.

Process 6 (80 MB):

- Partition 2 (170 MB) is the first big enough left → allocate 80 MB there.

BEST-FIT: We look for the smallest partition that can hold each process:

Process 1 (200 MB):

- Partitions that can hold 200 MB are #4 (205 MB) and #5 (300 MB) and #6 (185 MB is too small).
- The smallest big enough is 205 MB → allocate there.

Process 2 (15 MB):

- Free partitions are #1 (100 MB), #2 (170 MB), #3 (40 MB), leftover 5 MB in #4, #5 (300 MB), #6 (185 MB).
- The smallest that can hold 15 MB is partition 3 (40 MB) → allocate 15 MB there.

Process 3 (185 MB):

- Free partitions are #1 (100 MB), #2 (170 MB), leftover 25 MB in #3, leftover 5 MB in #4, #5 (300 MB), #6 (185 MB).
- The smallest that can hold 185 MB is #6 (exactly 185 MB) → allocate there.

Process 4 (75 MB):

- Free partitions are #1 (100 MB), #2 (170 MB), leftover 25 MB in #3, leftover 5 MB in #4, #5 (300 MB).
- The smallest that can hold 75 MB is #1 (100 MB) → allocate there.

Process 5 (175 MB):

- Free partitions are leftover 25 MB in #1, #2 (170 MB is too small), leftover 25 MB in #3, leftover 5 MB in #4, #5 (300 MB).
- The smallest that can hold 175 MB is #5 (300 MB) → allocate 175 MB there.

Process 6 (80 MB):

- Free partitions are leftover 125 MB in #5, #2 (170 MB), leftover 25 MB in #1, leftover 25 MB in #3, leftover 5 MB in #4.
- The smallest that can hold 80 MB is leftover 125 MB in #5 → allocate there.

WORST-FIT: We look for the largest partition that can hold each process

Process 1 (200 MB):

- The largest of [100, 170, 40, 205, 300, 185] is 300 MB → allocate 200 MB there.

Process 2 (15 MB):

- Remaining free partitions: 100, 170, 40, 205, leftover 100 in the (former) 300, 185.
- Largest is 205 → allocate 15 MB there, leaving 190 MB leftover in that partition.

Process 3 (185 MB):

- Remaining free partitions: 100, 170, 40, leftover 190, leftover 100, 185.
- The largest is leftover 190 → allocate 185 MB there, leaving 5 MB leftover.

Process 4 (75 MB):

- Remaining free partitions are 100, 170, 40, leftover 5, leftover 100, 185.
- The largest is 185 → allocate 75 MB, leaving 110 MB leftover.

Process 5 (175 MB):

- Remaining free partitions: 100, 170, 40, leftover 5, leftover 100, leftover 110.
- The largest is 170 or leftover 110 or leftover 100 or 100 or 40 or 5.
- 170 is not enough for 175, so no fit.
- Thus, process 5 cannot be allocated

We still try process 6 (80 MB):

- The largest free partition is 170 → it is big enough for 80, leaving 90 leftover. So process 6 is allocated, but process 5 is not satisfied.

Efficiency:

- First-Fit: Simple and fast to implement. However it leaves small leftover partitions near the beginning, this can lead to process not finishing in the different example however it passed all in this case.
- Best-Fit: Minimizes leftover space each time by using the smallest suitable memory, however it can create many small leftover memory scattered. In this example, it also places all processes successfully.
- Worst-Fit: Tries to leave behind the largest leftover memory by always using the largest available partition, however it fails to accommodate large processes later if it splits big memory in an inefficient manor. Here, it fails to place the 175 MB process.

5. Given: • 24-bit virtual address • 20-bit physical address • 4-KB page size (4 KB = 2^12 bytes)

    Conventional, single-level page table:

The total virtual address space is 2^24 bytes. Each page is 2^12 bytes.

The number of virtual pages = 2^(24 - 12) = 2^12 = 4096. A single-level page table has one entry per virtual page, so it has 4096 entries.

Inverted page table

The physical address space is 2^20 bytes. Each page is 2^12 bytes.

The number of physical frames = 2^(20 - 12) = 2^8 = 256. An inverted page table has one entry per physical frame, so it has 256 entries.

Maximum amount of physical memory

With a 20-bit physical address, the maximum physical memory is 2^20 bytes = 1 MB.

6. a) Without TLB: You need two memory references for each data access. One to look up the page table entry in memory and one to access the actual data in memory. Each memory reference is 50 ns, so the total is 50 + 50 = 100 ns.

   b) With TLB:
   On a TLB hit (75% of the time): TLB lookup = 2 ns. Data memory access = 50 ns. Total time = 2 + 50 = 52 ns.

   On a TLB miss (25% of the time): TLB lookup (miss) = 2 ns. Memory reference to get page table entry = 50 ns. Memory reference to get data = 50 ns. Total time = 2 + 50 + 50 = 102 ns.

   Effective Access Time (EAT) = (Hit Ratio × Hit Time) + (Miss Ratio × Miss Time) = (0.75 × 52) + (0.25 × 102) = 39 + 25.5 = 64.5 ns.

7. Paging is done so that the page table itself can be broken into smaller, separate pages rather than having to be one large, contiguous table in memory. This has two major benefits, It reduces memory usage as you only need to allocate space for those parts of the page table that are actually in use. As well it simplifies managing large address spaces by avoiding the need for one huge page table. Instead, the page table is itself paged, so we allocate and load only the relevant portions into memory.

8. f

FIFO REPLACEMENT: Track pages in the order they were loaded (oldest gets replaced first).

Step	Reference	Action & Explanation	Frames After Step	FIFO Order (Oldest → Newest)	Fault?
1	7	Memory empty; load 7.	[7, –, –]	[7]	Fault
2	2	2 not present; load it.	[7, 2, –]	[7, 2]	Fault
3	3	3 not present; load it	[7, 2, 3]	[7, 2, 3]	Fault
4	1	1 not present; frames are full so remove the 7 which is oldest load 1.	[1, 2, 3]	[2, 3, 1]	Fault
5	2	2 is already in memory.	[1, 2, 3]	[2, 3, 1]	–
6	5	5 not present; remove oldest in queue and load 5.	[1, 5, 3]	[3, 1, 5]	Fault
7	3	3 is in memory.	[1, 5, 3]	[3, 1, 5]	–
8	4	4 not present; remove oldest and load 4.	[1, 5, 4]	[1, 5, 4]	Fault
9	6	6 not present; remove oldest and load 6.	[6, 5, 4]	[5, 4, 6]	Fault
10	7	7 not present; remove oldest and load 7.	[6, 7, 4]	[4, 6, 7]	Fault
11	7	7 is in memory.	[6, 7, 4]	[4, 6, 7]	–
12	1	1 not present; remove oldest and load 1.	[6, 7, 1]	[6, 7, 1]	Fault
13	0	0 not present; remove oldest and load 0.	[0, 7, 1]	[7, 1, 0]	Fault
14	5	5 not present; remove oldest and load 5.	[0, 5, 1]	[1, 0, 5]	Fault
15	4	4 not present; remove oldest and load 4.	[0, 5, 4]	[0, 5, 4]	Fault
16	6	6 not present; remove oldest and load 6.	[6, 5, 4]	[5, 4, 6]	Fault
17	2	2 not present; remove oldest and load 2.	[6, 2, 4]	[4, 6, 2]	Fault
18	3	3 not present; remove oldest and load 3.	[6, 2, 3]	[6, 2, 3]	Fault
19	0	0 not present; remove oldest and load 0.	[0, 2, 3]	[2, 3, 0]	Fault
20	1	1 not present; remove oldest and load 1.	[0, 1, 3]	[3, 0, 1]	Fault

Total FIFO Page Faults: 17

LRU REPLACEMENT: Replace the page that was used least recently.

Step	Reference	Action & Explanation	Frames After Step	LRU Order (Oldest → Newest)	Fault?
1	7	Memory empty; load 7.	[7, –, –]	[7]	Fault
2	2	2 not present; load it.	[7, 2, –]	[7, 2]	Fault
3	3	3 not present; load it.	[7, 2, 3]	[7, 2, 3]	Fault
4	1	1 not present; frames full so remove the LRU page and load 1.	[1, 2, 3]	[2, 3, 1]	Fault
5	2	2 is in memory; reorder recency	[1, 2, 3]	[3, 1, 2]	–
6	5	5 not present; remove the LRU page and load 5.	[1, 2, 5]	[1, 2, 5]	Fault
7	3	3 not present; frames full so remove the LRU page and load 3.	[3, 2, 5]	[2, 5, 3]	Fault
8	4	4 not present; remove the LRU page and load 4.	[3, 4, 5]	[3, 5, 4]	Fault
9	6	6 not present; remove the LRU page and load 6.	[6, 4, 5]	[4, 5, 6]	Fault
10	7	7 not present; remove the LRU page and load 7.	[6, 7, 5]	[5, 6, 7]	Fault
11	7	7 is in memory; reorder recency	[6, 7, 5]	[5, 6, 7]	–
12	1	1 not present; remove the LRU page and load 1.	[6, 7, 1]	[6, 7, 1]	Fault
13	0	0 not present; remove the LRU page and load 0.	[0, 7, 1]	[7, 1, 0]	Fault
14	5	5 not present; remove the LRU page and load 5.	[0, 5, 1]	[1, 0, 5]	Fault
15	4	4 not present; remove the LRU page and load 4.	[0, 5, 4]	[0, 5, 4]	Fault
16	6	6 not present; remove the LRU page and load 6.	[6, 5, 4]	[5, 4, 6]	Fault
17	2	2 not present; remove the LRU page and load 2.	[6, 2, 4]	[4, 6, 2]	Fault
18	3	3 not present; remove the LRU page and load 3.	[6, 2, 3]	[6, 2, 3]	Fault
19	0	0 not present; remove the LRU page and load 0.	[0, 2, 3]	[2, 3, 0]	Fault
20	1	1 not present; remove the LRU page and load 1.	[0, 1, 3]	[3, 0, 1]	Fault

LRU total page faults = 18

OPTIMAL REPLACEMENT: Replace the page that will not be used for the longest time in the future.

Step	Reference	Action & Explanation	Frames After Step	(No Ongoing Order)	Fault?
1	7	Memory empty; load 7.	[7, –, –]	–	Fault
2	2	2 not present; load it.	[7, 2, –]	–	Fault
3	3	3 not present; load it.	[7, 2, 3]	–	Fault
4	1	1 not present; frames full; remove page not used for the longest time and load 1.	[1, 2, 3]	–	Fault
5	2	2 is in memory; no replacement needed.	[1, 2, 3]	–	–
6	5	5 not present; remove page not used for the longest time and load 5.	[1, 5, 3]	–	Fault
7	3	3 is in memory; no replacement needed.	[1, 5, 3]	–	–
8	4	4 not present; remove page not used for the longest time and load 4.	[1, 5, 4]	–	Fault
9	6	6 not present; remove page not used for the longest time and load 6.	[1, 5, 6]	–	Fault
10	7	7 not present; remove page not used for the longest time and load 7.	[1, 5, 7]	–	Fault
11	7	7 is in memory; no replacement needed.	[1, 5, 7]	–	–
12	1	1 is in memory; no replacement needed.	[1, 5, 7]	–	–
13	0	0 not present; remove page not used again and load 0.	[1, 5, 0]	–	Fault
14	5	5 is in memory; no replacement needed.	[1, 5, 0]	–	–
15	4	4 not present; remove page not used again and load 4.	[1, 4, 0]	–	Fault
16	6	6 not present; remove page not used again and load 6.	[1, 6, 0]	–	Fault
17	2	2 not present; remove page not used again and load 2.	[1, 2, 0]	–	Fault

Step	Reference	Action & Explanation	Frames After Step	(No Ongoing Order)	Fault?
18	3	3 not present; remove page not used again and load 3.	[1, 3, 0]	–	Fault
19	0	0 is in memory; no replacement needed.	[1, 3, 0]	–	–
20	1	1 is in memory; no replacement needed.	[1, 3, 0]	–	–

Optimal total page faults = 13

Final Results:

LRU: 18 page faults, FIFO: 17 page faults, Optimal: 13 page faults

9. Copy-on-Write is a technique that allows parent and child processes initially to share the same pages in memory marked as read-only. If one process attempts to modify a shared page, a page fault occurs, and the operating system then creates a private copy of that page for the process. This approach is beneficial when a forked child process does not end up modifying large portions of its address space. Hardware support is required, and the system must support read-only page protection, also must raise a page-fault exception if a process attempts to write to a read-only page, and the operating system then handles the fault by copying the page.

10. Maximum Acceptable Page-Fault Rate:

Memory-access time = 100 ns

If a page fault occurs and the replaced page is not modified, or an empty frame is available → 8 ms = 8×10^6 ns

If the replaced page is modified (70% of the time) → 20 ms = 2×10^7 ns

Effective Access Time (EAT) must be no more than 200 ns

Average page-fault service time (PF_time): PF_time = 0.3 × 8 ms + 0.7 × 20 ms = 0.3 × 8×10^6 ns + 0.7 × 2×10^7 ns = 2.4×10^6 ns + 14×10^6 ns = 16.4×10^6 ns

EAT = (1 – p) × 100 ns + p × (PF_time + 100 ns)

p ≤ 100 / (16.4×10^6)

Hence, the maximum acceptable page-fault rate is approximately 6.1×10^–6 (about 0.00061%)