CS5600 Homework 03
Memory Paging

Hang Zhao, NUID: 002826538

February 12, 2026

Problem 1 [30 points]: States of Memory

A simplified view of thread states is Ready, Running, and Blocked, when a thread is either ready and waiting to be scheduled, is running on the processor, or is blocked (for example, waiting for I/O).

Assuming a thread is in the Running state, answer the following questions, and explain your answer:

(a) Will the thread change state if it incurs a page fault? If so, to what state will it change?

Yes, the thread will move from Running to **Blocked**.

When a page fault happens, it means the page the thread needs isn't in physical memory right now – it's sitting on disk somewhere. So the OS has to go fetch that page from disk, and disk I/O is slow. The thread can't keep running while we wait for the disk, so the OS blocks it. Once the page finally gets loaded into a frame in memory, the OS moves the thread to Ready, and then eventually it gets scheduled back to Running.

So the state transition is: Running → Blocked (waiting for disk I/O) → Ready (page loaded) → Running (scheduled again).

(b) Will the thread change state if it generates a TLB miss that is resolved in the page table? If so, to what state will it change?

No, the thread stays in **Running**.

A TLB miss just means the translation wasn't cached in the TLB. But the key part of this question is "resolved in the page table" – that means the page IS in physical memory, we just didn't have the mapping cached. The hardware (or OS, depending on the architecture) walks the page table, finds the valid entry, updates the TLB, and the thread keeps going. This whole thing happens pretty fast and doesn't require any disk I/O, so there's no reason to block or deschedule the thread. It's basically a small delay but not a state change.

(c) Will the thread change state if an address reference is resolved in the page table? If so, to what state will it change?

No, the thread stays in **Running**.

This is pretty much the same situation as part (b). If the address gets resolved in the page table, that means the page is already in memory. The page table has a valid mapping for it. Whether we got here through a TLB miss or whatever, the point is we found what we needed without going to disk. No I/O needed means no blocking, so the thread just continues running. The MMU handles the translation and the thread doesn't even notice anything happened.

Problem 2 [30 points]: Page Faults

Consider the following page reference string:

7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0, 1

Assuming demand paging with three frames, how many page faults would occur for the following replacement algorithms?

(a) LRU replacement

With LRU, we always evict the page that was used least recently. I'll trace through every reference step by step. A * means page fault.

Step	Ref	Frame 1	Frame 2	Frame 3
1	7*	7		
2	2*	7	2	
3	3*	7	2	3
4	1*	1	2	3
5	2	1	2	3
6	5*	1	2	5
7	3*	3	2	5
8	4*	3	4	5
9	6*	3	4	6
10	7*	7	4	6
11	7	7	4	6
12	1*	7	1	6
13	0*	7	1	0
14	5*	5	1	0
15	4*	5	4	0
16	6*	5	4	6
17	2*	2	4	6
18	3*	2	3	6
19	0*	2	3	0
20	1*	1	3	0

Let me walk through some of the trickier steps:

- Step 4: Frames are [7, 2, 3]. Need page 1. LRU is 7 (used at step 1, oldest). Evict 7, load 1.
- Step 6: Frames are [1, 2, 3]. Need page 5. LRU is 3 (used at step 3). Wait – 2 was used at step 5, 1 at step 4, 3 at step 3. So evict 3. Load 5.
- Step 7: Frames are [1, 2, 5]. Need page 3. LRU is 1 (step 4). Evict 1. Load 3.

- Step 8: Frames are [3, 2, 5]. Need page 4. LRU is 2 (step 5). Evict 2. Load 4.
- Step 9: Frames are [3, 4, 5]. Need page 6. LRU is 5 (step 6). Evict 5. Load 6.
- Step 10: Frames are [3, 4, 6]. Need page 7. LRU is 3 (step 7). Evict 3. Load 7.

Total page faults: 18

(b) FIFO replacement

With FIFO, we evict whichever page was loaded first (oldest in memory). I track the load order using a queue.

Step	Ref	Frame 1	Frame 2	Frame 3	Queue (oldest→newest)
1	7*	7			7
2	2*	7	2		7, 2
3	3*	7	2	3	7, 2, 3
4	1*	1	2	3	2, 3, 1
5	2	1	2	3	2, 3, 1
6	5*	1	5	3	3, 1, 5
7	3	1	5	3	3, 1, 5
8	4*	1	5	4	1, 5, 4
9	6*	6	5	4	5, 4, 6
10	7*	6	7	4	4, 6, 7
11	7	6	7	4	4, 6, 7
12	1*	6	7	1	6, 7, 1
13	0*	0	7	1	7, 1, 0
14	5*	0	5	1	1, 0, 5
15	4*	0	5	4	0, 5, 4
16	6*	6	5	4	5, 4, 6
17	2*	6	2	4	4, 6, 2
18	3*	6	2	3	6, 2, 3
19	0*	0	2	3	2, 3, 0
20	1*	0	1	3	3, 0, 1

Some key steps:

- Step 5: Page 2 is already in memory, so no fault. FIFO queue doesn't change (2 doesn't move to back – that's LRU, not FIFO).
- Step 7: Page 3 is already in memory, no fault. Same idea.
- Step 8: Need page 4. Oldest in queue is 3. Evict 3, load 4.

Total page faults: 17

(c) Optimal replacement

With Optimal, we evict the page that won't be used for the longest time in the future (or never used again).

Step	Ref	Frame 1	Frame 2	Frame 3	Eviction reason
1	7*	7			–
2	2*	7	2		–
3	3*	7	2	3	–
4	1*	1	2	3	7: next use at step 10 (farthest)
5	2	1	2	3	–
6	5*	1	2	5	3: next use at step 18 (farthest)
7	3*	1	3	5	2: next use at step 17 (farthest)
8	4*	1	3	4	5: next use at step 14 (farthest)
9	6*	1	3	6	4: next use at step 15 (farthest)
10	7*	7	3	6	1: next use at step 12 (farthest)
11	7	7	3	6	–
12	1*	7	1	6	3: next use at step 18 (farthest)
13	0*	0	1	6	7: next use = never (farthest)
14	5*	0	5	6	1: next use at step 20 (farthest)
15	4*	0	5	4	6: next use at step 16 vs 0 at 19 vs 5 at never...
16	6*	6	5	4	0: next use at step 19 (farthest)
17	2*	6	2	4	5: next use = never
18	3*	6	2	3	4: next use = never
19	0*	0	2	3	6: next use = never
20	1*	0	1	3	2: next use = never

Even with the optimal algorithm, we get a lot of faults here because the reference string has so many different pages (0–7) and we only have 3 frames. There's just not enough room.

Total page faults: 15

Problem 3 [40 points]: Page Replacement Algorithms

(a) Define a page-replacement algorithm using the counter-based idea.

Here's how I'd define the algorithm:

i. Initial value of counters: Each page frame starts with a counter of 0. When a page is first loaded into a frame, its counter is set to 1 (since one page is now associated with that frame).

ii. When are counters increased? The counter for a frame is incremented by 1 every time the page in that frame is referenced (hit). So if a page keeps getting accessed, its counter goes up, meaning it's heavily used.

iii. When are counters decreased? Counters are decremented by 1 whenever a different page is loaded into any frame (i.e., on a page fault, all OTHER frames' counters get decremented by 1, but not below 0). This way, pages that haven't been used recently will have their counters decay over time.

iv. How is the replacement page selected? When a page fault occurs and all frames are full, we pick the frame with the smallest counter value. If there's a tie, we pick the one that was loaded earliest (FIFO as tiebreaker). The idea is that the smallest counter means that page has been used the least, so it's the best candidate to evict.

(b) Page faults for the reference string with four page frames

Reference string: 1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2

I'll trace through using my algorithm. Each frame has a counter. On a hit, that frame's counter goes up by 1. On a fault, all other frames' counters go down by 1 (min 0), and the frame with the smallest counter gets replaced.

Step	Ref	F1 (ctr)	F2 (ctr)	F3 (ctr)	F4 (ctr)	Fault?
1	1	1(1)				Yes
2	2	1(0)	2(1)			Yes
3	3	1(0)	2(0)	3(1)		Yes
4	4	1(0)	2(0)	3(0)	4(1)	Yes
5	5	5(1)	2(0)	3(0)	4(0)	Yes, evict 1(ctr=0, oldest)
6	3	5(1)	2(0)	3(1)	4(0)	No, hit
7	4	5(1)	2(0)	3(1)	4(1)	No, hit
8	1	5(0)	1(1)	3(0)	4(0)	Yes, evict 2(ctr=0, oldest)
9	6	5(0)	1(0)	6(1)	4(0)	Yes, evict 3(ctr=0)
10	7	5(0)	1(0)	6(0)	7(1)	Yes, evict 4(ctr=0)
11	8	8(1)	1(0)	6(0)	7(0)	Yes, evict 5(ctr=0)
12	7	8(1)	1(0)	6(0)	7(1)	No, hit
13	8	8(2)	1(0)	6(0)	7(1)	No, hit
14	9	8(1)	9(1)	6(0)	7(0)	Yes, evict 1(ctr=0)
15	7	8(1)	9(1)	6(0)	7(1)	No, hit
16	8	8(2)	9(1)	6(0)	7(1)	No, hit
17	9	8(2)	9(2)	6(0)	7(1)	No, hit
18	5	8(1)	9(1)	5(1)	7(0)	Yes, evict 6(ctr=0)
19	4	8(0)	9(0)	5(0)	4(1)	Yes, evict 7(ctr=0)
20	5	8(0)	9(0)	5(1)	4(1)	No, hit
21	4	8(0)	9(0)	5(1)	4(2)	No, hit
22	2	2(1)	9(0)	5(0)	4(1)	Yes, evict 8(ctr=0, tie w/ 9)

Total page faults: 13
(Steps 1–5, 8–11, 14, 18, 19, 22 are faults.)

(c) Minimum page faults using optimal replacement with four frames

For optimal replacement, on each fault we look ahead in the reference string and evict the page that won't be needed for the longest time.

Reference string: 1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2

Step	Ref	F1	F2	F3	F4	Eviction reason
1	1*	1				–
2	2*	1	2			–
3	3*	1	2	3		–
4	4*	1	2	3	4	–
5	5*	1	5	3	4	2: next use = step 22 (farthest)
6	3	1	5	3	4	hit
7	4	1	5	3	4	hit
8	1	1	5	3	4	hit
9	6*	1	5	6	4	3: next use = never (farthest)
10	7*	1	5	6	7	4: next use = step 19 (farthest)
11	8*	1	8	6	7	5: next use = step 18 vs 6 never... evict 6
		1	8	5	7	Actually let me redo: evict 6 (never used again)
11	8*	1	8	5	7	evict 6: next use = never
12	7	1	8	5	7	hit
13	8	1	8	5	7	hit
14	9*	9	8	5	7	evict 1: next use = never
15	7	9	8	5	7	hit
16	8	9	8	5	7	hit
17	9	9	8	5	7	hit
18	5	9	8	5	7	hit
19	4*	4	8	5	7	evict 9: next use = never
20	5	4	8	5	7	hit
21	4	4	8	5	7	hit
22	2*	4	2	5	7	evict 8: next use = never

Counting the faults: steps 1, 2, 3, 4, 5, 9, 10, 11, 14, 19, 22.
Minimum page faults with optimal replacement: 11