Lab 9: Memory Management

Giancarlo Succi, Leonard Johard

Innopolis University
Course of Operating Systems

2023

Virtual memory – Recap

- How to translate the virtual address to physical address?

Virtual memory – Recap

- How to translate the virtual address to physical address?
- What do we store in the page table?

Virtual memory – Recap

- How to translate the virtual address to physical address?
- What do we store in the page table?
- When does a page fault occur?

Virtual memory – Recap

- How to translate the virtual address to physical address?
- What do we store in the page table?
- When does a page fault occur?
- What are page-in and page-out operations?

Virtual memory – Recap

- How to translate the virtual address to physical address?
- What do we store in the page table?
- When does a page fault occur?
- What are page-in and page-out operations?
- What does the page offset represent?

Virtual memory – Recap

- How to translate the virtual address to physical address?
- What do we store in the page table?
- When does a page fault occur?
- What are page-in and page-out operations?
- What does the page offset represent?
- What is demand paging?

What is page replacement?

- When a page that is residing in virtual memory is requested by a process, the Operating System needs to decide which page will be replaced by this requested page.

Why do we need page replacement?

- Since the actual RAM is much less than the virtual memory, page faults occur. So whenever a page fault occurs, the Operating system has to replace an existing page in RAM with the newly requested page. In this scenario, page replacement algorithms help the Operating System in deciding which page to replace. The primary objective of all the page replacement algorithms is to minimize the number of page faults.

Page replacement algorithms

- Page Replacement Algorithms decide which page to remove when a new page needs to be loaded into the main memory. Page Replacement happens when a requested page is not present in the main memory and the available space is not sufficient for allocation to the requested page.
- A page replacement algorithm tries to select which pages should be replaced so as to minimize the total number of page misses. There are many different page replacement algorithms. These algorithms are evaluated by running them on a particular string of memory reference and computing the number of page faults. The fewer is the page faults the better is the algorithm for that situation.
- Here, we will discuss the following page replacement algorithms:
  - FIFO algorithm
  - LRU algorithm
  - NFU algorithm
  - Aging algorithm

First In First Out (FIFO) algorithm

- This is the simplest page replacement algorithm. In this algorithm, the OS maintains a queue that keeps track of all the pages in memory, with the oldest page at the front and the most recent page at the back.
- When there is a need for page replacement, the FIFO algorithm, swaps out the page at the front of the queue, that is the page which has been in the memory for the longest time.

FIFO page replacement algorithm

Example: Consider the page reference string of size 12: 1, 2, 3, 4, 5, 1, 3, 1, 6, 3, 2, 3 with $F = 4$ frames and $P = 6$ pages. Using FIFO page replacement algorithm, calculate the number of hits/misses:

1	2	3	4	5	1	3	1	6	3	2	3

1	1	1	1	5	5	5	5	5	5	2	2
	2	2	2	2	1	1	1	1	1	1	1
		3	3	3	3	3	3	6	6	6	6
			4	4	4	4	4	4	3	3	3
M	M	M	M	M	M	H	H	M	M	M	H

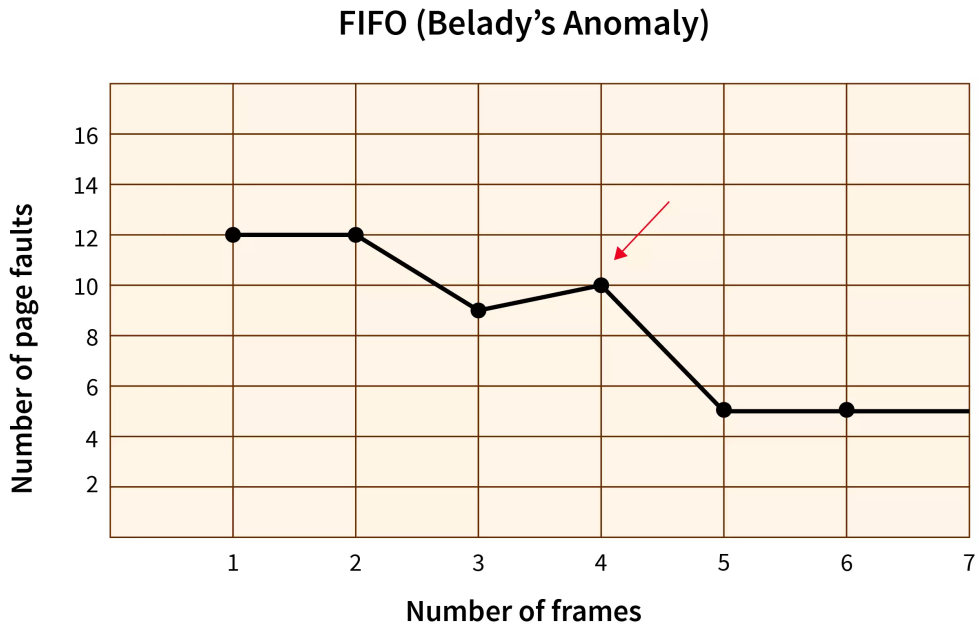
M = Miss
H = Hit

Total Page Faults = number of misses = 9
number of hits = 12-9=3

FIFO page replacement algorithm

- Advantages
  - Simple and easy to implement.
- Disadvantages
  - Poor performance.
  - Does not consider the frequency of page usage or last used time. It simply replaces the oldest page.
  - Suffers from Belady's Anomaly(i.e. more page faults when we increase the number of page frames).
- **Belady's Anomaly** refers to the counterintuitive situation in virtual memory management where increasing the number of page frames results in an increase in the number of page faults.
- In *FIFO* algorithm, the number of page faults increases even if we increase the number of frames. The pages that are added first to the queue will be replaced first, so the oldest pages will be swapped first.
- In stack-based page replacement algorithms, Belady's Anomaly does not occur.

FIFO page replacement algorithm



In the above figure, the page fault is the highest when the number of frames is 1. After that, as the number of frames increases, the number of faults is the same up to frame number 2. At point 3, the number of faults decreased. But, at point 4, the number of faults again increased even if the frames are increased. At this point, we can clearly see **"Belady's Anomaly"**.

Least Recently Used (LRU) algorithm

- This algorithm keeps track of page usage over a short period of time. It works on the idea that the pages that have been most heavily used in the past are most likely to be used heavily in the future too.
- In LRU, whenever page replacement happens, the page which has not been used for the longest amount of time is replaced.

Least Recently Used (LRU) algorithm

Example: Consider the page reference string of size 12: 1, 2, 3, 4, 5, 1, 3, 1, 6, 3, 2, 3 with $F = 4$ frames and $P = 6$ pages. Using LRU page replacement algorithm, calculate the number of hits/misses:

1	2	3	4	5	1	3	1	6	3	2	3
1	1	1	1	5	5	5	5	5	5	2	2
	2	2	2	2	1	1	1	1	1	1	1
		3	3	3	3	3	3	3	3	3	3
			4	4	4	4	4	6	6	6	6
M	M	M	M	M	M	H	H	M	H	M	H

M = Miss
H = Hit

Total Page Faults = number of misses = 8
number of hits = 12 - 8 = 4

Not Frequently Used (NFU) algorithm

- Associate a **counter** with each page
- On every timer interrupt, the OS looks at each page. If the Referenced Bit is set then it increments that page's counter clears the bit.
- The counter approximates how often the page is used.
- For replacement, choose the page with lowest counter.
- **Problems:**
  - Some pages may be heavily used
    - Their counter is large
  - The program's behavior changes. Now, this page is not used ever again (or only rarely)
  - This algorithm *never forgets!* This page will never be chosen for replacement!

Ageing algorithm

- Associate a **counter** with each page
- On every timer interrupt, the OS looks at each page. If the Referenced Bit is set then it **shifts referenced bits into counters of pages**, then it clears those bits.
- The counter approximates **an age for the page which decreases when the page is not referenced**.
- For replacement, choose the page with lowest counter.
- It is clear that a page that has not been referenced for, say, four clock ticks will have four leading zeros in its counter and thus will have a lower value than a counter that has not been referenced for three clock ticks.

Aging algorithm

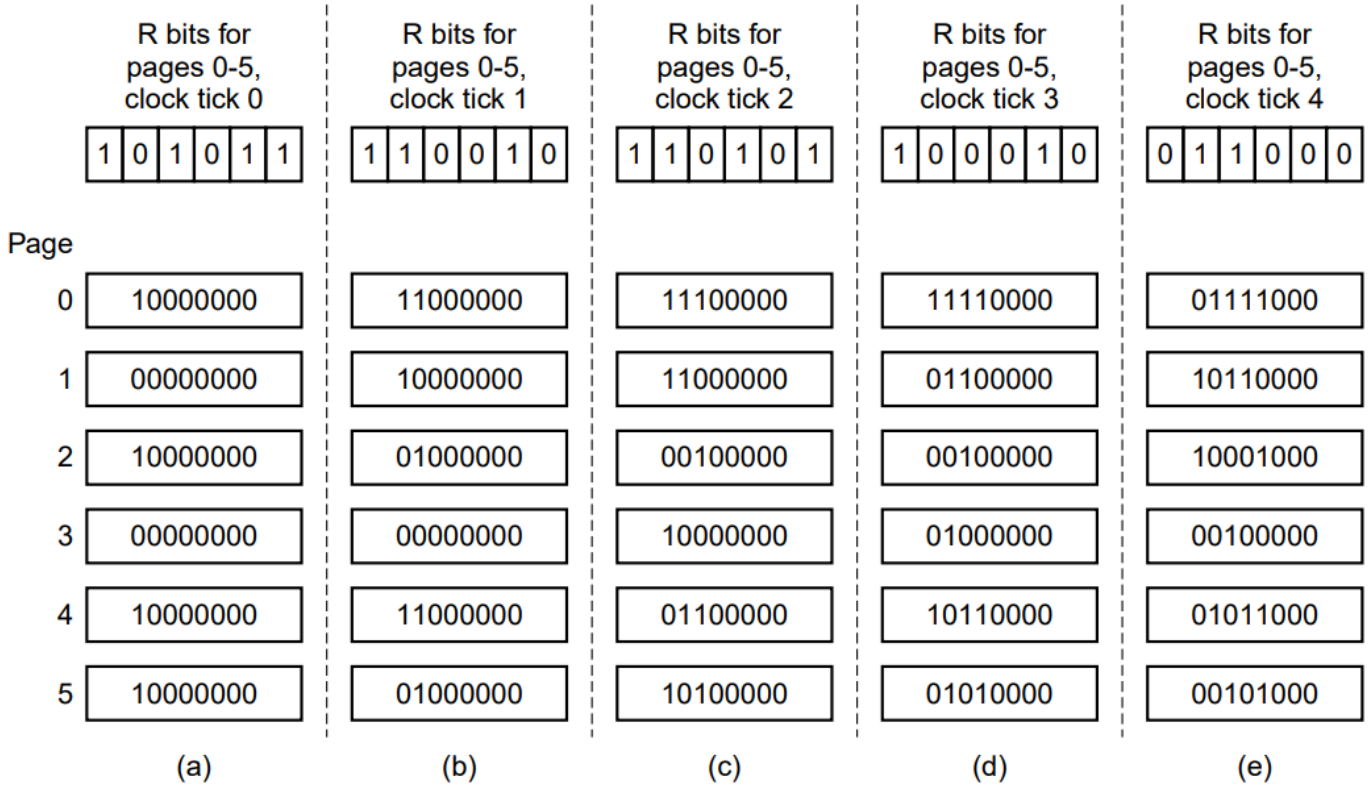


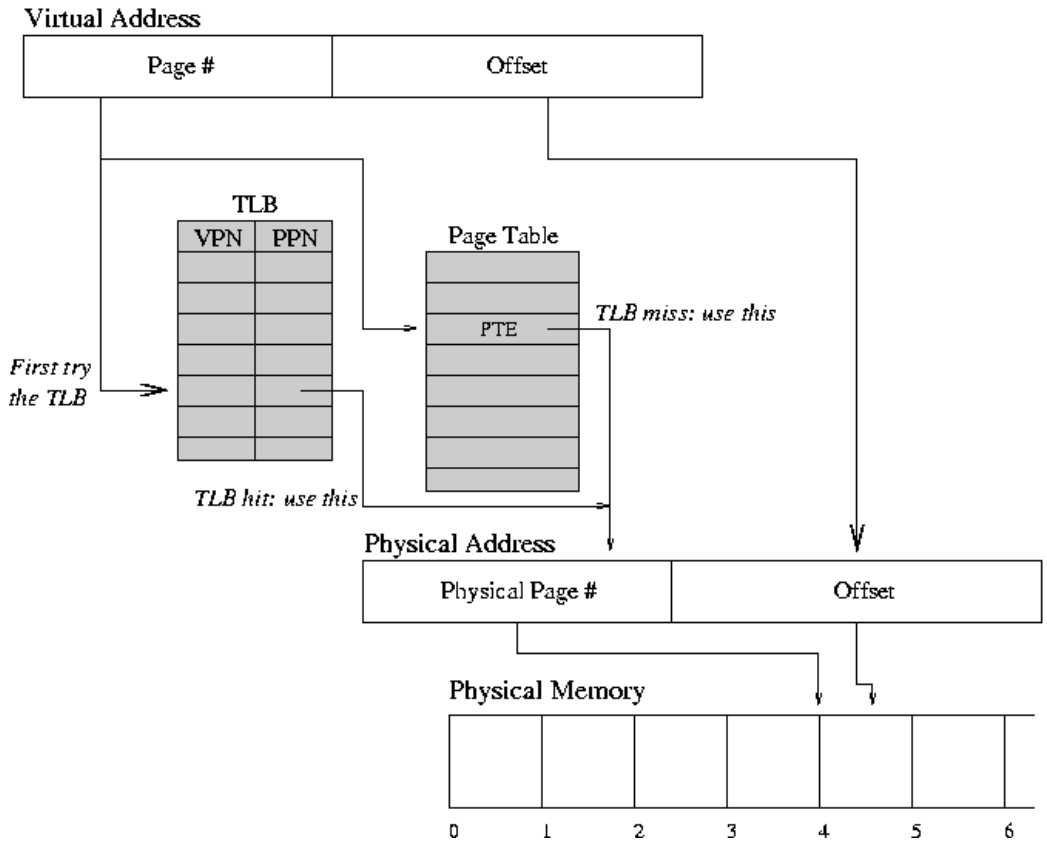
Figure 4-4. The aging algorithm simulates LRU in software. Shown are six pages for five clock ticks. The five clock ticks are represented by (a) to (e).

Aging algorithm

- This algorithm differs from LRU in two ways.
  - Consider pages 3 and 5 in Fig. 4-4(e). Neither has been referenced for two clock ticks; both were referenced in the tick prior to that. According to LRU, if a page must be replaced, we should choose one of these two. The trouble is, we do not know which of these two was referenced last in the interval between tick 1 and tick 2. By recording only one bit per time interval, we have lost the ability to distinguish references early in the clock interval from those occurring later. All we can do is remove page 3, because page 5 was also referenced two ticks earlier and page 3 was not.
  - The second difference between LRU and aging is that in aging the counters have a finite number of bits, 8 bits in this example. Suppose that two pages each have a counter value of 0. All we can do is pick one of them at random. In practice, however, 8 bits is generally enough if a clock tick is around 20 msec. If a page has not been referenced in 160 msec, it probably is not that important.

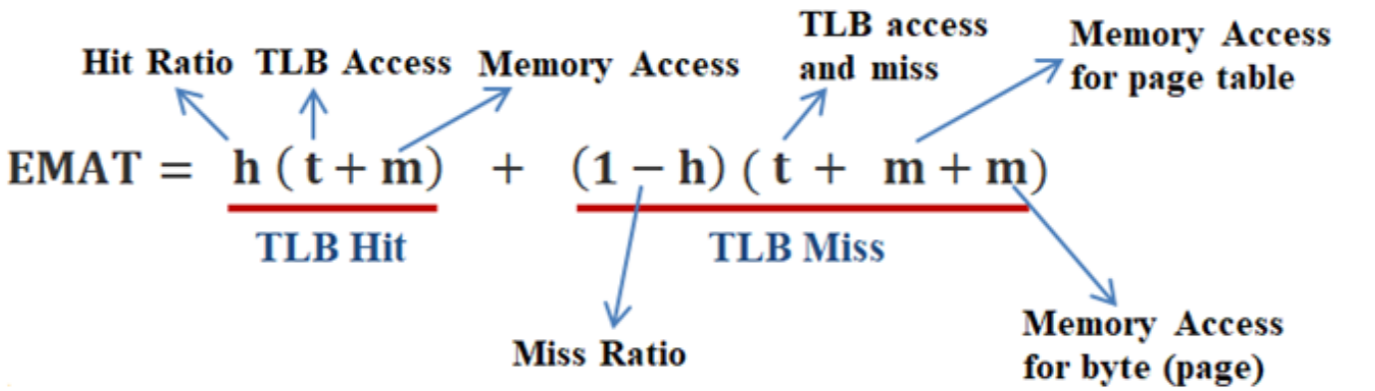
Translation Lookaside Buffer (TLB)

TLB is a type of memory cache that stores recent translations of virtual memory to physical addresses to enable faster retrieval. This high-speed cache is set up to keep track of recently used page table entries (PTEs).



Paging performance

- The performance of paging depends on various factors, such as page size, page replacement algorithms, page table size,...etc
- Translation lookaside buffer(TLB) is added to improve the performance of paging. TLB access time will be very less compared to the main memory access time.
- Some performance metrics of paging.
  - Miss ratio = $\frac{misses}{\text{total references}}$
  - Hit ratio = $\frac{hits}{\text{total references}} = 1 - \text{Miss ratio}$
  - Effective memory access time with TLB (without page faults) is:



t=TLB access time, m=main memory access time, h=TLB hit

Paging performance

- Effective memory access time with TLB (including page faults) is:

$$EMAT = pf\_rate*$$

$$(EMAT(without\_page\_faults) + page\_fault\_service\_time)$$

$$+(1 - pf\_rate) * EMAT(without\_page\_faults)$$

`pf_rate = page fault rate`

The time taken to service the page fault is called as **page fault service time**.

Page replacement algorithms

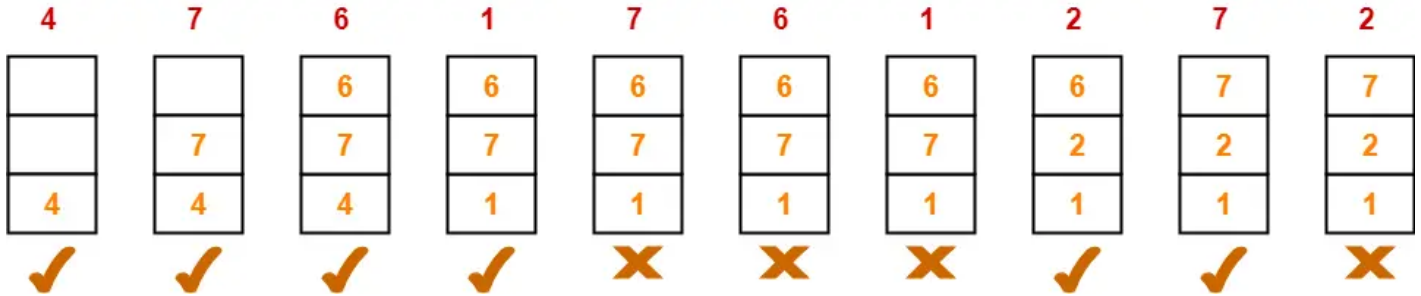
Example:

- A system uses 3 page frames for storing process pages in main memory. It uses the Least Recently Used (LRU) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below: 4 , 7, 6, 1, 7, 6, 1, 2, 7, 2 Also calculate the hit ratio and miss ratio.

Page replacement algorithms

Example:

- A system uses 3 page frames for storing process pages in main memory. It uses the Least Recently Used (LRU) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below: 4 , 7, 6, 1, 7, 6, 1, 2, 7, 2 Also calculate the hit ratio and miss ratio.
- Total number of references = 10.



Total number of page faults occurred = 6. Hit ratio = 0.4 or 40%. Miss ratio = 1 - Hit ratio = 0.6 or 60%

TLB

Example:

- Consider a single level paging scheme with a TLB. Assume no page fault occurs. It takes 20 ns to search the TLB and 100 ns to access the physical memory. If TLB hit ratio is 80%. Calculate the TLB miss ratio and the effective memory access time (EMAT).

TLB

Example:

- Consider a single level paging scheme with a TLB. Assume no page fault occurs. It takes 20 ns to search the TLB and 100 ns to access the physical memory. If TLB hit ratio is 80%. Calculate the TLB miss ratio and the effective memory access time (EMAT).
- **Solution:**
  - Number of levels of page table = 1
  - TLB access time = 20 ns
  - Main memory access time = 100 ns
  - TLB Hit ratio = 80% = 0.8
  - TLB Miss ratio = 1 – TLB Hit ratio = 1 – 0.8 = 0.2
  - Effective Access Time (EMAT) = 0.8 x (20 ns + 100 ns) + 0.2 x (20 ns + 2 x 100 ns) = 0.8 x 120 ns + 0.2 * 220 ns = 96 ns + 44 ns = 140 ns.
  - Thus, effective memory access time = 140 ns.

Exercise 1 (1/3)

- In this exercise, You need to extend the implementation of ex2 from lab 08.
- You have to implement three functions as follows:

```
// Random page replacement
int random(struct PTE* page_table);
// NFU page replacement
int nfu(struct PTE* page_table);
// Aging page replacement
int aging(struct PTE* page_table);
```

Note: You can add more arguments to the functions if needed but all of them should have the same signature. The return value of the functions should be the page number to evict which is selected based on the algorithm, e.g. **nfu** function should select the page to evict based on the NFU page replacement algorithm.

Exercise 1 (2/3)

- The program **pager.c** should accept the page replacement algorithm as a command line argument (in addition to the number of frames and pages). The user can pass "random" for Random page replacement algorithm, "nfu" for NFU page replacement algorithm, or "aging" for Aging page replacement algorithm.
- The program **pager.c** should print an info message to the user about the selected algorithm.
- Extend the source code **mmu.c** to include a benchmarking functionality. The program **mmu.c** should calculate and print to stdout the *hit ratio* after finishing all requests considering that hit occurs when MMU gets a request to a valid page whereas miss occurs when it gets a request to an invalid page.

Exercise 1 (3/3)

- Run the program three times, one time for each algorithm using the same input. Compare the performance of the algorithms and add your findings to **ex1.txt**.
- A sample input for the program can be found in this gist.
- Submit **mmu.c**, **pager.c** and **ex1.txt** and a script **ex1.sh** to run your programs.
- **Note:** For Aging algorithms, assume that:
  - there is a timer interrupt for each page access request.
  - the counters are unsigned chars (8-bit integers) and each counter has n bits where $n = sizeof(\text{unsigned char})$. For NFU, assume that the the counter of type integer.
- To shift the variable num to the left x bits, we can write: `num << x` which equals to $num = num * (2^x)$.
- To shift the variable num to the right x bits, we can write: `num >> x` which equals to $num = num/(2^x)$.
- You can add referenced bit (r_bit) to the counter (8 bits) as follows: `counter = (counter >> 1) | (r_bit << 7)`

Exercise 2 (Bonus exercise)

- In this exercise, You need to extend the implementation of ex2 from lab 08. You have to add a TLB functionality to **mmu.c** program. The TLB here is represented as an array of **struct TLB_entry** which has only two fields as follows:

  ```
  struct TLB_entry {int page, frame;};
  ```

- In TLB, you need to store the most recent mappings between pages and frames. If you cannot find the requested page in TLB, then there is a TLB miss and you need to check the page table.
- We assume that the size of TLB is 20% size of the page table.
- Your program should calculate the TLB miss ratio and print it to stdout.
- Try different values for number of frames and pages. Compare the results and add the findings to **ex2.txt**.
- Submit **mmu.c**, **pager.c** and **ex2.txt**.
- Submit also a script **ex2.sh** to run your programs.

References

- Andrew S. Tanenbaum and Herbert Bos. 2014. Modern Operating Systems (4th. ed.). Prentice Hall Press, USA.
- https://www.scaler.com/
- https://afteracademy.com/blog/what-are-the-page-replacement-algorithms
- https://mycareerwise.com/content/effective-access-time-using-hit-miss-ratio/content/exam/gate/computer-science

End of lab 9