CS4411 Operating Systems Homework 2 (Memory) Solutions Spring 2019

Due on Tuesday, April 23, 2019 before class

1. **[10 points]** Consider the following segment table:

Segment	*Base*	*Length*
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

What are the physical addresses for each of the following logical addresses?

- 1, 20
- 3, 450
- 6, 90

All values are in decimal.

Answer:

- $1,20$:
  The segment number and offset of $(1,20)$ are 1 and 20, respectively. Because the offset 20 is larger than length of segment 1 (*i.e.*, 14), this is an invalid address.
- $3,450$:
  This address refers to segment 3 and offset 450. Segment 3 has its base address 1327 and length 580. Because offset $450 < 580$, this is a valid address and the physical address is $1777 = 1327\text{base} + 450$.
- $6,90$:
  This address refers to segment 6 and offset 90. Segment 6 has no entry in the segment table, and hence this is an invalid address.

■

2. **[10 points]** Consider the following page table. All numbers in the table are binary.

Page Number	*Frame Number*
0000	1101
0001	0011
0010	0110
0011	0000
0100	1101

- Given the physical address of each of the following logical addresses. Assume that addresses occupy 12 bits.
  - 001100000010_2 (base 2)
  - 400_{10} (base 10)
  - 000001100111_2 (base 2)
- What is the page size in this system?

Answer:

- 001100000010_2:
  The logical address 001100000010_2 is divided into a 4-bit page number and a 8-bit offset. Thus, 001100000010_2 has its 4-bit page number 0011_2 and offset 00000010_2. Because the page number 0011_2 has its page frame in frame 0000_2, the physical address is 000000000010_2, where blue and red indicate page frame number and offset, respectively.

- 400_{10}:
  Note that $400_{10} = 110010000_2$. Therefore, page number and offset are 0001_2 and 10010000_2, respectively. Page 0001_2 has its page frame number 0011_2, and the corresponding physical address is 001110010000_2, where blue and red indicate page frame number and offset, respectively. The corresponding physical address in decimal is 912_{10}.
- 000001100111_2:
  The logical address 001100000010_2 is divided into a 4-bit page number and a 8-bit offset. Thus, 000001100111_2 has its 4-bit page number 0000_2 and offset 01100111_2. Because the page number 0000_2 has its page frame in frame 1101_2, the physical address is 110101100111_2, where blue and red indicate page frame number and offset, respectively.

∎

3. **[15 points]** Suppose we have a page trace $\{4,3,2,1,4,3,5,4,3,2,1,5\}$. Use FIFO, LRU and MIN (the optimal algorithm) to run this page trace with 3 page frames and then with 4 page frames, and report the following:

   - For each page replacement algorithm, use two tables to show the page replacement activities, one table for 3 page frames and the other for 4 page frames.
   - For each table, clearly show the number of page faults, hit ratio, and miss ratio.
   - For each page replacement algorithm, indicate Belady anomaly if there is any.
   - In each table, circle the pages that were brought into physical memory due to page faults.
   - The memory content may or may not change after each page reference. Compare the 3-frame table and the 4-frame page, verify that after each page reference whether the 4-frame column contains the 3-frame column in terms of the pages on that column. This is to verify the "inclusion property". For each column, circle the page in the 4-frame table that is **not** in the 3-frame table.

   Note that near the end of the MIN (optimal) algorithm, there could be multiple choice of pages to be evicted. To ensure we will speak the same language, if you have a multiple choice, **always select the page with the smallest number to evict.** **Use the table format used in our class slides.**

   Answer: The following uses the convention shown below. This is the same as needed in Problem (4) to save my time.

   - Each column shows the content *after* referring to the page shown at the top.
   - Pages that caused page faults are in boldface with asterisks.
   - For FIFO, the columns are ordered based on the time pages are loaded into physical memory.
   - For LRU, the columns are ordered based on last reference time. If a **page fault occurs**, the new page is "pushed" on to the column and other pages on that column are ordered based on the time each page was used last time. As a result, the least recently used page is at the bottom of that column.

     If there is **no page fault**, meaning that the used page is in memory, then the used page is moved to the top and all pages between the top and the original position of the used page are "pushed down".
   - The MIN algorithm is similar to the LRU. If **a page fault occurs**, the new page is pushed to the top of a column and the remaining pages are ordered based on the next use time. Therefore, the page that will not be used for the longest period of time is at the bottom or evicted.

     If **there is no page fault**, this means the used page is in memory. In this case, we use the same strategy as discussed in the LRU case. More precisely, the used page is moved to the top, and all pages between the original position of the used page and the top are "pushed down" and the order of these pages is adjusted based on the next use time.

   Question: Why is this "re-ordering" not needed for LRU?

   The following is the FIFO results. On each column, pages are ordered based on the time each page was loaded into physical memory, with the earliest at the bottom.

Sequence	1	2	3	4	5	6	7	8	9	10	11	12
Page	4	3	2	1	4	3	5	4	3	2	1	5
Frame 1	$\mathbf{4}^*$	$\mathbf{3}^*$	$\mathbf{2}^*$	$\mathbf{1}^*$	$\mathbf{4}^*$	$\mathbf{3}^*$	$\mathbf{5}^*$	5	5	$\mathbf{2}^*$	$\mathbf{1}^*$	1
Frame 2		4	3	2	1	4	3	3	3	5	2	2
Frame 3			4	3	2	1	4	4	4	3	5	5
Page Faults = 9 Miss Ratio = 9/12 = 75% Hit Ratio = 3/12 = 25%												

Sequence	1	2	3	4	5	6	7	8	9	10	11	12
Page	4	3	2	1	4	3	5	4	3	2	1	5
Frame 1	$\mathbf{4}^*$	$\mathbf{3}^*$	$\mathbf{2}^*$	$\mathbf{1}^*$	1	1	$\mathbf{5}^*$	$\mathbf{4}^*$	$\mathbf{3}^*$	$\mathbf{2}^*$	$\mathbf{1}^*$	$\mathbf{5}^*$
Frame 2		4	3	2	2	2	1	5	4	3	2	1
Frame 3			4	3	3	3	2	1	5	4	3	2
Frame 4				4	4	4	3	2	1	5	4	3
Page Faults = 10 Miss Ratio = 10/12 = 83% Hit Ratio = 2/12 = 17%												

Two observations are immediate. **First**, it shows the existence of Belady anomaly: the 4-frame case has higher number of page faults than the 3-frame case. **Second**, the inclusion property is not satisfied. Consider $p_7 = 5$. In the 3-frame case, after $p_7 = 5$ the pages in physical memory are $\{5,3,4\}$ while in the 4-frame case, the pages in physical memory are $\{5,1,2,3\}$. The case of $p_8 = 4$ exhibits the same result.

The following shows the results of LRU. Pages in memory are ordered by the time from now to each page's previous reference time.

Sequence	1	2	3	4	5	6	7	8	9	10	11	12
Page	4	3	2	1	4	3	5	4	3	2	1	5
Frame 1	$\mathbf{4}^*$	$\mathbf{3}^*$	$\mathbf{2}^*$	$\mathbf{1}^*$	$\mathbf{4}^*$	$\mathbf{3}^*$	$\mathbf{5}^*$	4	3	$\mathbf{2}^*$	$\mathbf{1}^*$	$\mathbf{5}^*$
Frame 2		4	3	2	1	4	3	5	4	3	2	1
Frame 3			4	3	2	1	4	3	5	4	3	2
Page Faults = 10 Miss Ratio = 10/12 = 83% Hit Ratio = 2/12 = 17%												

Sequence	1	2	3	4	5	6	7	8	9	10	11	12
Page	4	3	2	1	4	3	5	4	3	2	1	5
Frame 1	$\mathbf{4}^*$	$\mathbf{3}^*$	$\mathbf{2}^*$	$\mathbf{1}^*$	4	3	$\mathbf{5}^*$	4	3	$\mathbf{2}^*$	$\mathbf{1}^*$	$\mathbf{5}^*$
Frame 2		4	3	2	1	4	3	5	4	3	2	1
Frame 3			4	3	2	1	4	3	5	4	3	2
Frame 4				4	3	2	1	1	1	5	4	3
Page Faults = 8 Miss Ratio = 8/12 = 67% Hit Ratio = 4/12 = 33%												

Note that $p_8 = 4$ in the 3-frame case is in memory, page 4 is brought to the top and the remaining pages 5 and 3 are "pushed down". $p_9 = 3$ in the 3-frame case, and $p_5 = 4$, $p_6 = 3$ and $p_8 = 4$ and $p_9 = 3$ in the 4-frame case are processed the same way. See slide 32 of `09-Virtual-memory.pdf` for the details.

It is clear that for each page reference the 4-frame result contains the 3-frame result (*i.e.*, inclusion property). It is even more important to observe that the extra frame in the 4-frame results always contains the least recently used pages, which is evicted in the 3-frame case.

The following is the results of the optimal MIN algorithm. The subscripts are "forward" distances, the distance from now to the next use. An ∞ indicates the page is no more needed. Note that we agreed that we evict the page with the smallest page number if there is a multiple choice, and, consequently, the page with ∞ and the smallest page number is at the bottom for every page order adjustment.

Sequence	1	2	3	4	5	6	7	8	9	10	11	12
Page	4	3	2	1	4	3	5	4	3	2	1	5
Frame 1	$\mathbf{4}^*_4$	$\mathbf{3}^*_4$	$\mathbf{2}^*_7$	$\mathbf{1}^*_7$	4_3	3_3	$\mathbf{5}^*_5$	4_∞	3_∞	$\mathbf{2}^*_\infty$	$\mathbf{1}^*_\infty$	5_∞
Frame 2		4_3	4_2	4_1	1_6	4_2	4_1	5_4	5_3	5_2	5_1	1_∞
Frame 3			3_3	3_2	3_1	1_5	3_2	3_1	4_∞	4_∞	4_∞	4_∞
Page Faults = 7 Miss Ratio = 7/12 = 58% Hit Ratio = 5/12 = 42%												

Sequence	1	2	3	4	5	6	7	8	9	10	11	12
Page	4	3	2	1	4	3	5	4	3	2	1	5
Frame 1	$\mathbf{4}_4^*$	$\mathbf{3}_4^*$	$\mathbf{2}_7^*$	$\mathbf{1}_7^*$	4_3	3_3	$\mathbf{5}_5^*$	4_∞	3_∞	2_∞^*	$\mathbf{1}_\infty^*$	5_∞
Frame 2		4_3	4_2	4_1	1_6	4_2	4_1	5_4	5_3	5_2	5_1	1_∞
Frame 3			3_3	3_2	3_1	1_5	3_2	3_1	4_∞	4_∞	4_∞	4_∞
Frame 4				2_6	2_5	2_4	2_3	2_2	2_1	3_∞	3_∞	3_∞
Page Faults = 6 Miss Ratio = 6/12 = 50% Hit Ratio = 4/12 = 50%												

Let us take a look at a few reordering cases in the 4-frame table.

- In the case of $p_5 = 4$, because page 4 is in memory, it is moved to the top and page 1 is pushed down. Because there is only one page above page 4, no reorder is needed.
- The next page is $p_6 = 3$, which is in memory. Thus, page 3 is moved to the top, and the two pages above page 3 (*i.e.*, page 4 and page 1) are pushed down and reordered by its forward distance (actually no change of order).
- Next is $p_7 = 5$, which is not in memory. In this case, $p_7 = 5$ is pushed to the top and the other pages 3_3, 4_2, 1_5 (and 2_4 in the 4-frame case) are reordered by the recorded forward distance as $\{4_2, 3_3, 2_4, 1_5\}$, In the 3-frame case, because page 1 has the longest forward distance, it is evicted and the column has $\{5_5, 4_1, 3_2\}$. In the 4-frame case, the column becomes $\{5_5, 4_1, 3_2, 2_3\}$. Note the updated distances.

■

4. **[10 points]** In general, LRU and MIN have a lot in common. After each reference of a page, the used page is "pushed" on to the top of a column (*i.e.*, stack) and a portion or all pages on that column are re-ordered, depending on whether a page fault occurs or not. For each page in memory, except for the just used one, LRU orders the pages based on "backward distance", the distance from now to a page's previous use with the longest backward distance (*i.e.*, least recently used) page at the bottom, while MIN order the pages based on "forward distance", the distance from now to a page's next use with the longest forward distance (*i.e.*, not used for the longest period of time) at the bottom.

If a **page fault occurs**, the re-ordering applies to **all** pages on a column. On the other hand, if **no page fault occurs** (*i.e.*, the used page being in memory), the ordering only applies to all pages above the used page. These pages are pushed down one position so that the used page can be pushed to the top. Because of these stack-like operations, LRU and MIN are two examples of the **stack algorithms** for page replacement.

Let us take a look at a LRU example. Suppose the column (or stack) currently has pages 7 (top), 3, 4 and 1 (bottom). If the next page reference is page 4, then 4 is moved to the top and the pages between 4 and the top (*i.e.*, 7 and 3) are pushed down. The new column (or stack) becomes 4 (top, most recently used), 7, 3, 1. Page 1 is not moved because it is below the newly used page 4. However, if the newly used page is 5, which is not in memory, page 5 is brought in and takes the top position, and all pages are pushed down. The result is 5, 7, 3, 4 and 1; however, if the memory only has 4 page frames, the result is 5, 7, 3, and 4, and page 1 is evicted.

Now, let us turn to the MIN algorithm. We use 4_3 (top), 1_6, 3_1 and 2_5 (bottom) as an example, where the subscripts are the forward distances. More precisely, 4_3 indicates page 4 has a forward distance 3, meaning page 4 will be used 3 time units later; 1_6 indicates page 1 has a forward distance 6, meaning page 1 will be used 6 time units later; 3_1 indicates page 3 has a forward distance 1, meaning page 3 will be used next; and 2_5 indicates page 2 has a forward distance 5, meaning page 2 will be used 4 time units later. The next page reference is page 3 (because its forward distance is 1), page 3 is moved to the top and its forward distance receives a new value, say 3 (*i.e.*, 3_3). The pages between the top and page 3 (*i.e.*, pages 4_3 and 1_6) are pushed down and reordered based on their updated forward distance. Hence, the new column (or stack) becomes 3_3, 4_2, 1_5 and 2_4. What if the next page is page 5? Because page 5 is not in memory, it is brought into memory. Because the current pages in memory are 3_3, 4_2, 1_5 and 2_4 with page 1 having the largest forward distance 5, page 1 is evicted, and the new column becomes 5_5, 4_1, 3_2 and 2_3, where the subscript 5 in 5_5 indicates that page 5 will be used 5 time units later.

Problem (3) has a page trace of 12 page references. Run that page trace with 3 page frames, 4 page frame and 5 page frames. However, pages on each column in the table must be reordered in the indicated way.

Do the following problems:

- Trace the page trace using LRU and MIN with 3 frames, 4 frames and 5 frames.
- Combined the 3-frame, 4-frame and 5-frame into a single table of 5 rows such that the first 3 rows correspond to the 3-frame results, the first 4 rows correspond to the 4-frame results, and the first 5 rows correspond to the 5-frame results. Is this doable?
- If it can be done, what you can find from this? Present a detailed discussion.

This problem is related to Problem (6).

Answer: Because Problem (3) has done the 3-frame and 4-frame cases in the described way, we only show the 5-frame case here.

The following is the results for the case of 5 frames. Two thick lines divides the 3-frame and 4-frame cases. As you can see for each page reference p_t, the content of 5-frame column contains the content of 4-frame column, which, in turn, contains the content of 3-frame column. Therefore, LRU satisfies the inclusion property. Of course, this is **NOT** a proof; it is just an example.

Sequence	1	2	3	4	5	6	7	8	9	10	11	12
Page	4	3	2	1	4	3	5	4	3	2	1	5
Frame 1	**4***	**3***	**2***	**1***	4	3	**5***	4	3	2	1	5
Frame 2		4	3	2	1	4	3	5	4	3	2	1
Frame 3			4	3	2	1	4	3	5	4	3	2
Frame 4				4	3	2	1	1	1	5	4	3
Frame 5							2	2	2	1	5	4
Page Faults = 5 Miss Ratio = 5/12 = 42% Hit Ratio = 7/12 = 58%												

The following is the 5-frame result using MIN. Again, it is clear that the inclusion property is satisfied.

Sequence	1	2	3	4	5	6	7	8	9	10	11	12
Page	4	3	2	1	4	3	5	4	3	2	1	5
Frame 1	$\mathbf{4}_4^*$	$\mathbf{3}_4^*$	$\mathbf{2}_7^*$	$\mathbf{1}_7^*$	4_3	3_3	$\mathbf{5}_5^*$	4_∞	3_∞	2_∞	1_∞	5_∞
Frame 2		4_3	4_2	4_1	1_6	4_2	4_1	5_4	5_3	5_2	5_1	1_∞
Frame 3			3_3	3_2	3_1	1_5	3_2	3_1	4_∞	4_∞	4_∞	4_∞
Frame 4				2_6	2_5	2_4	2_3	2_2	2_1	3_∞	3_∞	3_∞
Frame 5							1_4	1_3	1_2	1_1	2_∞	2_∞
Page Faults = 5 Miss Ratio = 5/12 = 52% Hit Ratio = 7/12 = 58%												

One more important observation has to be made from the above two tables: The extra row in the 5-frame case contains the evicted pages in the 4-frame case, and the extra row in the 4-frame case contains the evicted pages in the 3-frame case. **Why?** For each page fault, a new page is brought in and a page is evicted. For LRU, the least recently used page is at the bottom of the stack, and the new page that is pushed on to the stack pushes the bottom page out to the next row. This concept will be used to prove that LRU satisfies the inclusion property in Problem (6). The MIN algorithm just acts the same way. ∎

5. **[10 points]** A process has four page frames allocated to it. The table below includes the following items: (1) the time of the last loading of a page into each frame, (2) the time of last access to a page in each frame, (3) the virtual page number in each frame, and (4) the reference (R) and modified (M) bits for each frame. The times are in clock ticks, from the process starts at time zero to the event.

Virtual Page No.	*Page Frame*	*Time Loaded*	*Time Used*	*R*-bit	*M*-bit
2	0	60	150	0	1
1	1	130	160	0	0
0	2	70	155	1	0
3	3	100	145	1	1

A page fault to virtual page 4 has occurred. Which page frame will have its content replaced for each of the following page replacement algorithm: FIFO, LRU and Clock? Explain why in each case.

Answer:

- **FIFO**:
  The *Time Loaded* column shows that the oldest page is page 2, which was loaded into physical memory at time 60. Therefore, the page in page frame 0 (*i.e.*, page 2) should be evicted. Because it has the modified bit set, a page-out is needed.
- **LRU**:
  The *Time Used* column shows that page frame 3 has not been used for the longest period of time because the last use was at time 145. Therefore, LRU will pick the page in page frame 3, which is page 3. Because this page has its modified bit set, a page-out is needed.
- **Clock**:
  CASE I: Suppose the clock pointer is currently pointing to page frame 0. Because page frame 0 has a 0 referenced bit, its content (*i.e.*, page 2) has to be replaced. Because this page frame has the modified bit set, a page-out is needed.

  CASE II: Suppose the clock pointer is currently pointing to page frame 1. This is similar to the above case. But, no page-out is needed.

  CASE III: Suppose the clock pointer is currently pointing to page frame 2. Because page frame 2 has it referenced bit set, the clock algorithm resets it to 0 and moves to page frame 3. Because page frame 3 also has its referenced bit set, the clock algorithm resets it to 0 and moves to page frame 0. Once returning to page frame 0, because its referenced bit is not set, page 2 is the victim. A page-out is needed as in **CASE I**.

  CASE IV: Suppose the clock pointer is currently pointing to page frame 3. Because page frame 3 has it referenced bit set, the clock algorithm resets it to 0 and moves to page frame 0. Once returning to page frame 0, because its referenced bit is not set, page 2 is the victim. A page-out is needed as in **CASE I**.

■

6. **[10 points]** Let $P =< p_1, p_2, \ldots, p_n >$ be a page trace of size n and let m be the number of page frames. Also, let $M_t(P, \alpha, m)$ be the memory content (*i.e.*, pages) after referring to page p_t with respect to page replacement algorithm α. A page replacement algorithm satisfies the **Inclusion Property** if $M_t(P, \alpha, m) \subseteq M_t(P, \alpha, m+1)$ for every t. This expression suggests that the content in memory $M_t(P, \alpha, m)$ after referring to page p_t is a subset of $M_t(P, \alpha, m+1)$ that has one more page frame. In other words, for every page reference p_t, the content in memory with m page frames is a subset of the content in memory with $m+1$ page frames.

   Do the following problems:

   - Prove that if a page replacement algorithm satisfies the inclusion property, then Belady anomaly cannot happen!
   - Prove that the LRU algorithm does satisfy the inclusion property. (**Hint**: Recall the way of ordering the pages on a column based on how long a page was not used. (See Problem 4.) The most recently used page is at the top, while the least recently used page is at the bottom. When a page fault occurs, the bottom-most page is evicted (if there is no free page frames). Now, what would happen when each column has one more page frame! **This is not a difficult problem if you understand the concept of "stack algorithms".**

   If you are able to prove LRU satisfies the inclusion property, you should also be able to prove the optical algorithm MIN also satisfies the inclusion property.

   Answer: Let α be a page replacement algorithm, and let $M_t(P, \alpha, m)$ be the pages in memory of m page frames for page trace P after using page p_t. Because α satisfied the inclusion property, we have $M_t(P, \alpha, m) \subseteq M_t(P, \alpha, m+1)$ for every t and $\{p\} = M_t(P, \alpha, m+1) - M_t(P, \alpha, m)$ is the extra page in $M_t(P, \alpha, m+1)$ but not in $M_t(P, \alpha, m)$. If page q is used at time $t+1$, we have two cases to consider:

   - If $q \in M_t(P, \alpha, m)$, then there is no page fault. Because $M_t(P, \alpha, m) \subseteq M_t(P, \alpha, m+1)$, q is also in $M_t(P, \alpha, m+1)$ and hence no page fault will occur with $m+1$ page frames.
   - If $q \notin M_t(P, \alpha, m)$, a page fault occurs with m page frames. However, if $q = p$ (*i.e.*, the new page q being the extra page in $M_t(P, \alpha, m+1)$), even though there is a page fault with m page frames there is no page fault with $m+1$ page frames. If $q \neq p$, due to inclusion property $M_t(P, \alpha, m) \subseteq M_t(P, \alpha, m+1)$, q cannot be in $M_t(P, \alpha, m+1)$. In this case, the use of page q will cause page fault for both m page frames and $m+1$ page frames.

Therefore, if inclusion property is met, running with $m+1$ page frames will not generate more page faults than running with m page frames.

. . . .

Then, we shall prove that LRU satisfies the inclusion property using what we have learned in Problem (4). Suppose a process starts without pages in physical memory. As the execution continues, many pages are brought into physical memory until the m frames are fully occupied. Figure 1 below shows two memory configurations with m frames (left) and $m+1$ frames (right). Note that due to the way of filling the "stack" the content of the m-frame and $(m+1)$-frame cases are exactly the same. Note also that the $(m+1)$-frame case has an empty frame in yellow, while the m-frame case does not. So far, $M_t(P, \text{LRU}, m) \subseteq M_t(P, \text{LRU}, m+1)$ holds (because they are equal).

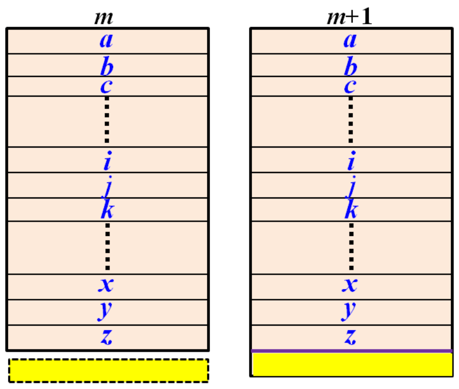


Figure 1: All Page Frames in the m-frame Configuration Are Filled.

Now consider the next page reference. We have two cases based on whether a page fault occurs or not.

CASE I – No Page Fault: This page is in both the m-frame and the $(m+1)$-frame. Assume that this page is page j. The LRU way of organizing the "stack" moves j to the top, and pages from a to i – the one above j – are pushed down (Figure 2). We still have $M_t(P, \text{LRU}, m) \subseteq M_t(P, \text{LRU}, m+1)$.

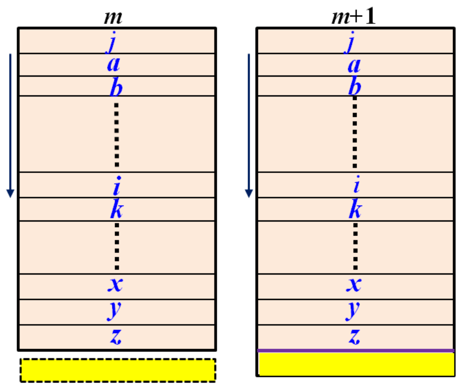


Figure 2: The Referenced Page is in m-frame, and Moved to the Top.

CASE II – Page Fault: If the next page reference p causes a page fault for the m-frame case. In this case, the new page p is pushed to the top of the stack and all other pages are pushed down. The least recently used page z is pushed out of memory. In the $(m+1)$-frame case, the least recently used page z is pushed to the bottom frame. Note that z is still in $(m+1)$-frame's memory, because $(m+1)$-frame has one more page frame (Figure 3).

What we have learned so far? Right after the m-frame memory is filled up, whether there is a page fault, $M_t(P, \text{LRU}, m)$ is a subset of $M_t(P, \text{LRU}, m+1)$. Note that the least recently used page in the $(m+1)$-frame case is the page getting "pushed" out in the m-frame case.

Now, you know the initial situation. Next, it is an induction stop. We assume after p_{t-1}, the $(t-1)^{\text{th}}$ page in page trace P, the $(m+1)$-frame case contains the content of the m-frame case (*i.e.*, $M_{t-1}(P, \text{LRU}, m) \subseteq$

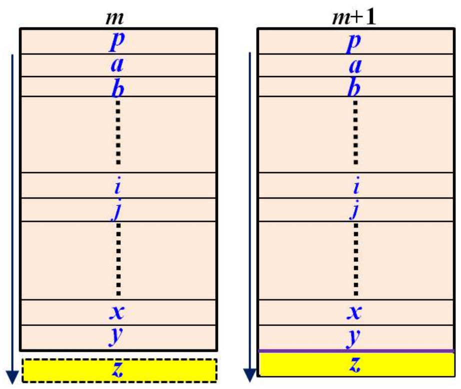


Figure 3: The Referenced Page is not in m-frame. All Pages in $(m+1)$-frame Are Pushed Down without a Page Fault.

$M_{t-1}(P,\text{LRU},m+1)$), and the extra page in the $(m+1)$-frame case contains the least recently used page of the m-frame case.

Suppose the next page is p_t. We have three cases to consider: **(1)** p_t is in $M_t(P,\text{LRU},m)$ and hence in $M_t(P,\text{LRU},m+1)$; **(2)** p_t is not in $M_t(P,\text{LRU},m)$ (*i.e.*, $p_t \notin M_t(P,\text{LRU},m)$) but is in $M_t(P,\text{LRU},m+1)$ (*i.e.*, $p_t \in M_t(P,\text{LRU},m+1)$) ; and **(3)** p_t is not in $M_t(P,\text{LRU},m+1)$ and hence not in $M_t(P,\text{LRU},m)$. If we can prove that $M_t(P,\text{LRU},m) \subseteq M_t(P,\text{LRU},m+1)$ holds, the inclusion property holds for LRU.

CASE I – Page $p_t \in M_t(P,\textbf{LRU},m)$:
In this case, $p_t \in M_{t-1}(P,\text{LRU},m) \subseteq M_{t-1}(P,\text{LRU},m+1)$, both the m-frame and $(m+1)$-frame cases will not have page fault. The stacks of $M_{t-1}(P,\text{LRU},m)$ and $M_{t-1}(P,\text{LRU},m+1)$ are only reorganized between page p_t and the top of the stack. As a result, if $M_{t-1}(P,\text{LRU},m) \subseteq M_{t-1}(P,\text{LRU},m+1)$, after p_t we still have $M_t(P,\text{LRU},m) \subseteq M_t(P,\text{LRU},m+1)$, and inclusion property is satisfied.

CASE II – Page $p_t \notin M_t(P,\textbf{LRU},m)$ but $p_t \in M_{t+1}(P,\textbf{LRU},m+1)$:
In this case, the m-frame case has a page fault, but the $(m+1)$-frame case does not. In the m-frame case, p_t is pushed to the top of the stack and all other page frames are pushed down. As a result, the least recently used page z is pushed out of the stack (Figure 4). Because $M_{t-1}(P,\text{LRU},m+1)$ contains $M_{t-1}(P,\text{LRU},m)$ and has only one more page, this extra page has to be p_t. Therefore, p_t is moved to the top and all other pages are pushed down, with z being the least recently used page. Again, we have $M_t(P,\text{LRU},m) \subseteq M_t(P,\text{LRU},m+1)$.

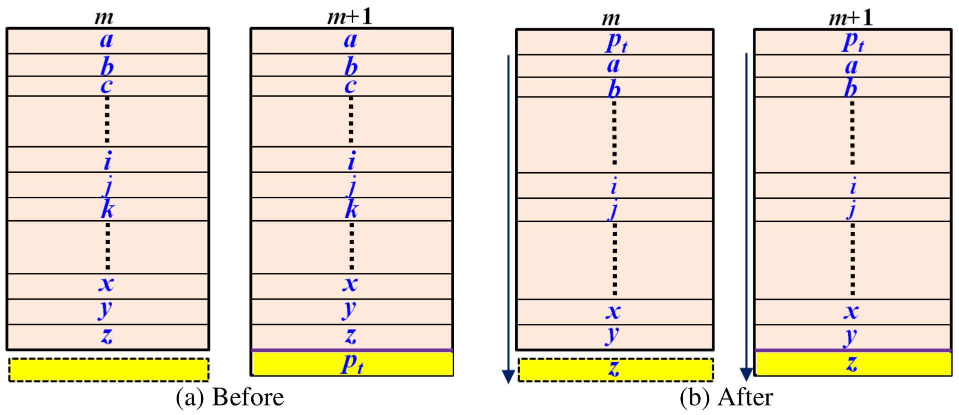


Figure 4: Page $p_t \notin M_{t-1}(P,\text{LRU},m)$ but $p_t \in M_{t-1}(P,\text{LRU},m+1)$

CASE III – Page $p_t \notin M_{t-1}(P,\textbf{LRU},m+1)$:
If the next page reference p_t is not in $M_{t-1}(P,\text{LRU},m+1)$, p_t cannot be in $M_{t-1}(P,\text{LRU},m)$. In this case, both m-frame and $(m+1)$-frame will have page fault. In the case of m-frame,p_t is pushed to the top of $M_t(P,\text{LRU},m)$ while all the existing pages are pushed down with the least recently used z out of $M_t(P,\text{LRU},m)$ (Figure 5). In the case of $(m+1)$-frame, its least recently used page Δ is at the bottom. Because p_t is pushed to the top of $M_t(P,\text{LRU},m+1)$, its least recently used page Δ is pushed out of $M_t(P,\text{LRU},m+1)$

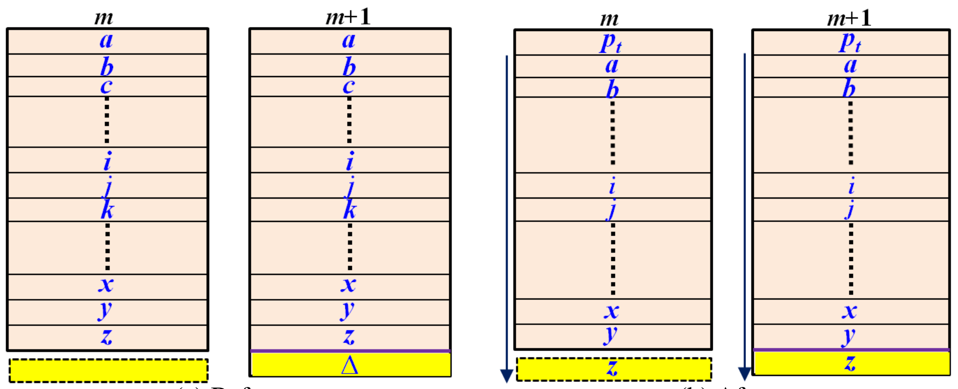


Figure 5: Page $p_t \notin M_{t-1}(P, \text{LRU}, m+1)$

Therefore, if $M_{t-1}(P, \text{LRU}, m) \subseteq M_{t-1}(P, \text{LRU}, m+1)$ holds, then $M_t(P, \text{LRU}, m) \subseteq M_t(P, \text{LRU}, m+1)$ also holds. Initially, before all page frames are filled up, both m-frame and $(m+1)$-frame have exactly the same pages. Once all page frames are filled up, the next page fault starts to satisfies inclusion property. From that point, inclusion property holds for every future page references. ■

7. **[10 points]** The working set (WS) of a process at virtual time t, written as $W(t, \theta)$, is the set of pages that were referenced in the interval $(t-\theta, t]$, where θ is the window size.[1] The problem with working set is that it is rather difficult to implement accurately. In 1972, Chu and Opderbeck proposed an interesting alternative to working set.[2] This is the Page Fault Frequency (PFF) replacement algorithm. Unlike WS which updates $W(t, \theta)$ after every page reference, PFF suggests that the set be updated only when a page fault occurs. More precisely, let t and t' be two consecutive page fault times, where $t' < t$. Then, the page that should be in physical memory at time t and window size θ $P(t, \theta)$ is defined as follows:

$$P(t,\theta) = \begin{cases} W(t, t-t'+1) & \text{if } t-t' > \theta \\ P(t',\theta) \cup \{p_t\} & \text{otherwise} \end{cases}$$

This means the following:

- If two consecutive page faults have a longer time span than the window size θ, then $P(t, \theta)$ is the working set from time t' to $t+1$.
- Otherwise, $P(t, \theta)$ is the older $P(t', \theta)$ plus the newly referenced page p_t.

The following is a table showing the PFF set for page trace $\{4,3,2,1,4,3,5,4,3,2,1,5\}$ and $\theta = 2$. In this table, bold face page numbers on the first row indicate page faults.

Time t	1	2	3	4	5	6	7	8	9	10	11	12
Page	4	3	2	1	4	3	5	4	3	2	1	5
	4	**3**	**2**	**1**	4	3	**5**	4	3	**2**	**1**	5
		4	3	2	1	4	3	5	4	3	2	1
			4	3	2	1	4	3	5	4	3	2
				4	3	2	1	1	1	5	4	3
											5	4
size	1	2	3	4	4	4	4	4	4	4	5	5

In this table, from $t = 1$ to $t = 4$ we have four page faults, and the time gap between two consecutive page faults is 1. Because these gaps are less than $\theta = 2$, $P(t, \theta)$ increases after every page fault. Note that the way of

[1]Peter J. Denning, The working set model for program behavior, *Communications of the ACM*, Vol. 11 (1968), No. 5, pp. 323-333 and Peter J. Denning, Working Set Past and Present, *IEEE Transactions on Software Engineering*, Vol. SE-6 (1980), No. 1, pp. 64–84.

[2]W. W. Chu and H. Opderbeck, The Page Fault Frequency Replacement Algorithm, *AFIPS Conference Proceedings*, 41 (1972 FJCC), pp. 597–609.

presenting this table resemble a stack on each column. On each column the order of the page numbers is based on the distance from now and the last reference. In this way, the most recently referenced page is at the top of that column (*i.e.*, stack). When a page is referenced, pages on that column are reordered. For example, after referring to page 1 at $t = 4$, the column has $\{1,2,3,4\}$. The next page is 4, which is at the bottom previously, moves to the top of the column for $t = 5$.

There is no page fault at time $t = 5$ and $t = 6$ and $P(5,\theta)$ and $P(6,\theta)$ do not change; but, page ordering is changed. At time $t = 7$ page $p_7 = 5$ causes a page fault. Because the last page fault was $t' = 4$ and the gap $t - t' = 7 - 4 = 3 > \theta = 2$, $P(7,\theta)$ should include all pages referenced between t' and t. Therefore, $P(t,\theta) = \{5,3,4,1\}$.

In the working set model, the number of pages in a working set is always no more than θ; however, in the PFF model the number of pages in $P(t,\theta)$ can be larger than θ as shown in the above table. Study the PFF algorithm, use PFF and WS to process the page trace with $\theta = 3$:

$$\{1,2,3,4,5,4,3,5,4,3,6,7,8,1,2,3,4,5,4,3,5,4,3,2,1,6,7,7,7,8\}$$

Now do the following:

- Use working set to process the above page trace with $\theta = 3$.
- Use PFF (Page Fault Frequency) algorithm to process the above page trace with $\theta = 3$.
- Draw a diagram with x-axis as the time (*i.e.*, 1, 2, 3, ...) and the y-axis as the number of pages in physical memory.

<u>**Answer**</u>: The following is the working set with $\theta = 3$.

Working Set with $\theta = 3$: Part I

Time t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Page	1	2	3	4	5	4	3	5	4	3	6	7	8	1	2
	$\mathbf{1}^*$	$\mathbf{2}^*$	$\mathbf{3}^*$	$\mathbf{4}^*$	$\mathbf{5}^*$	4	3	5	4	3	$\mathbf{6}^*$	$\mathbf{7}^*$	$\mathbf{8}^*$	$\mathbf{1}^*$	$\mathbf{2}^*$
		1	2	3	4	5	4	3	5	4	3	6	7	8	1
			1	2	3	3	5	4	3	5	4	3	6	7	8
size	1	2	3	3	3	3	3	3	3	3	3	3	3	3	3

Working Set with $\theta = 3$: Part II

Time t	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Page	3	4	5	4	3	5	4	3	2	1	6	7	7	7	8
	$\mathbf{3}^*$	$\mathbf{4}^*$	$\mathbf{5}^*$	4	$\mathbf{3}^*$	5	4	3	$\mathbf{2}^*$	$\mathbf{1}^*$	$\mathbf{6}^*$	$\mathbf{7}^*$	7	7	$\mathbf{8}^*$
	2	3	4	5	4	3	5	4	3	2	1	6	6		7
	1	2	3		5	4	3	5	4	3	2	1			
size	3	3	3	2	3	3	3	3	3	3	3	3	2	1	2

It is important to note that $W(19,\theta)$ has only 2 pages. Because $p_{19} = 4$, $p_{18} = 5$ and $p_{17} = 4$, there are only two different pages and hence $W(19,\theta) = \{4,5\}$. The same situation happens to $p_{28} = 7$, $p_{29} = 7$ and $p_{30} = 8$.

. . . .

The following is the PFF set with $\theta = 3$:

PFF with $\theta = 3$: Part I

Time t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Page	1	2	3	4	5	4	3	5	4	3	6	7	8	1	2
	$\mathbf{1}^*$	$\mathbf{2}^*$	$\mathbf{3}^*$	$\mathbf{4}^*$	$\mathbf{5}^*$	4	3	5	4	3	$\mathbf{6}^*$	$\mathbf{7}^*$	$\mathbf{8}^*$	$\mathbf{1}^*$	$\mathbf{2}^*$
		1	2	3	4	5	4	3	5	4	3	6	7	8	1
			1	2	3	3	5	4	3	5	4	3	6	7	8
				1	2	2	2	2	2	2	5	4	3	6	7
					1	1	1	1	1	1		5	4	3	6
													5	4	3
														5	4
															5
size	1	2	3	4	5	5	5	5	5	5	4	5	6	7	8

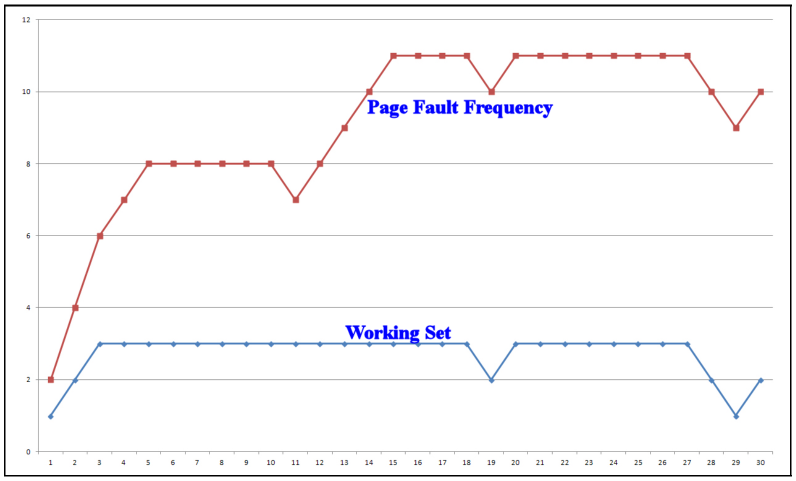
PFF with $\theta = 3$: Part II

Time t	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Page	3	4	5	4	3	5	4	3	2	1	6	7	7	7	8
	3	4	5	4	3	5	4	3	2	1	6	7	7	7	8
	2	3	4	5	4	3	5	4	3	2	1	6	6	6	7
	1	2	3	3	5	4	3	5	4	3	2	1	1	1	6
	8	1	2	2	2	2	2	2	5	4	3	2	2	2	1
	7	8	1	1	1	1	1	1	1	5	4	3	3	3	2
	6	7	8	8	8	8	8	8	8	8	5	4	4	4	3
	4	6	7	7	7	7	7	7	7	7	8	5	5	5	4
	5	5	6	6	6	6	6	6	6	6	7	8	8	8	5
size	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8

Note $p_{11} = 6$ that causes a page fault. The previous page fault is $p_5 = 5$. These two page faults has a distance of $11 - 5 = 6$, which is larger than $\theta = 3$. As a result, $P(11, \theta)$ should be the same as $W(11, 11 - 6 + 1) = W(11, 6) = \{3, 4, 5, 6\}$.

. . . .

Finally, the following is the size diagram:



As you may observe from the above diagram, PFF may have a very large number of pages in physical memory, while working set always maintains the size is no more than θ. Of course, PFF has a much smaller number of page faults, because it may have more pages in physical memory (*i.e.*, 8 vs. 3 in the above example). ∎