PAGE REPLACEMENT POLICIES SIMULATION

Team Name: Rangers

Project Title: Page Replacement Policies

Instructor/Mentor: Rishabh Malik

Member Details:

- Anjaneyulu Bairi 2021202008
- Praveen Bandaru 2021202012
- Sai Shushma Maturi 2021201012
- Sowmya Lahari Vajrala 2021202010

Introduction:

Paging is a memory management scheme that is used to retrieve processes from the secondary storage into the main memory in the form of pages. With this mechanism a large amount of memory can be provided to programmers on a smaller physical memory. The main memory is divided into frames and the processes are divided into pages. When a particular page of a process is required, it is first searched in the frames. If it is available, then it is used otherwise the required page will be fetched from secondary memory and will replace an already existing page in one of the frames. There are many algorithms to determine which page to be replaced in such cases. Whenever the required page is not found in main memory, it is termed as page fault. The main aim of any page replacement algorithm is to reduce the page fault rate.

Problem Description:

The objective of the project is to simulate various page replacement algorithms and do comparative analysis of the results. The simulator keeps track of the pages that are being loaded into the memory and the memory trace after each page reference. The number of misses or faults is recorded as well. For each reference string the algorithms mentioned below will be applied and the results are analyzed using graphs.

Algorithms:

- Random
- Optimal or MIN algorithm
- NRU (Not Recently Used)
- FIFO (First-In-First-Out)
- FIFO with second chance
- Clock
- LRU (Least Recently Used)

- NFU (Not Frequently Used)
- Working Set
- Aging (approximate LRU)
- WSClock

Approach:

1. **Random Page Replacement Algorithm:**

The Random page replacement algorithm replaces random page in memory. The main advantage of using random page replacement algorithms is it eliminates the overhead cost of tracking page references. Implementation of random page replacement algorithm is very easy compared to all other replacement algorithms. Generate a random number in the range of frame size and replace that page with a new page.

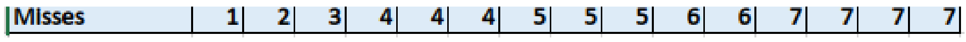
Working:

Reference string: 7 0 1 2 0 1 #1 3 2 #2 0 4 2 3 0 3 2

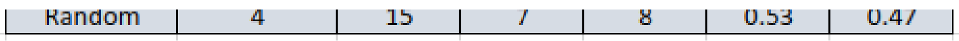
- # Represents that a page is being modified. As this algorithm does not use modified bit, these will be ignored.

Number of page frames: 4

Below is the memory trace:



Misses	1	2	3	4	4	4	5	5	5	6	6	7	7	7	7



Random	4	15	7	8	0.53	0.47

Summary:

Implementation of the random algorithm is very simple. If frames are full just select a random frame and replace it with a new page.

2. **Optimal or MIN algorithm:**

The optimal page replacement algorithm also known as MIN algorithm replaces a page that will not be referenced for a long time among the existing ones. This is impractical as the operating system will not know in advance which pages will be used in the future. This algorithm has the lowest page-fault rate among all the algorithms. So, despite its impracticality, this serves as a reference to compare the performance of other page replacement algorithms.

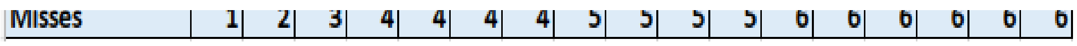
Working:

Reference string: 7 0 1 2 0 1 #1 3 2 #2 0 4 2 3 0 3 2

- # Represents that a page is being modified. As this algorithm does not use modified bit, these will be ignored.

Number of page frames: 4

Below is the memory trace



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

Summary:

Algorithm	Frame count	Ref count	Page faults	Hits	Hit ratio	Miss Ratio
Optimal	4	15	6	9	0.6	0.4

In the C++ implementation of Optimal page replacement algorithm, the main memory is simulated using a vector of length equal to number of frames. An unordered set is used to lookup if a page exists or not in the main memory. When all the frames are occupied and a page fault occurs, for all the pages in the main memory, the time at which they will be referenced in the future is extracted. Based on that the page that will be accessed later when compared to others will be replaced. The input is taken from a file and the output, which is the number of frames, reference string length, hit ratio and miss ratio are written to a file.

3. Not Recently Used (NRU)

In not recently used page replacement algorithm every page will have a referenced bit which indicates whether a page has been referenced at least one time after it is loaded and a modified bit which indicates whether a page has been modified once it is loaded. Based on these two bits the pages that are already present in the main memory are divided into 4 classes. They are given as follows:

Class 0: Not Referenced and Not Modified

Class 1: Not Referenced and Modified

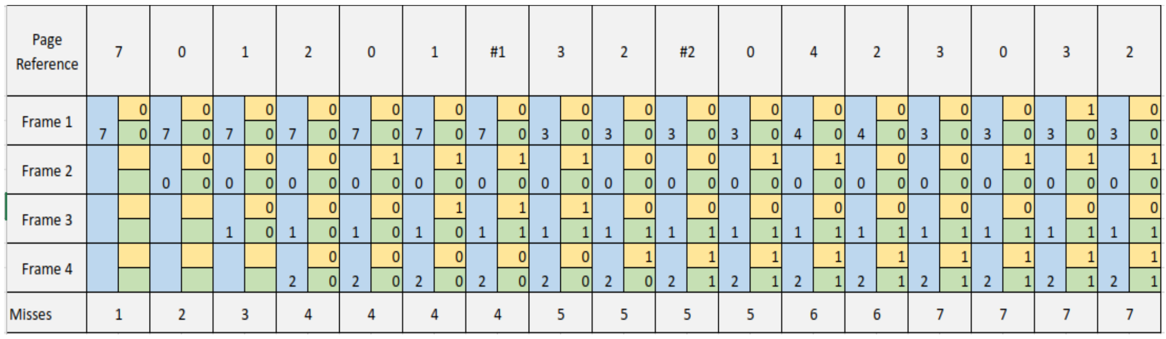
Class 2: Reference and Not Modified

Class 3: Referenced and Modified

After dividing the NRU picks a random page from the lowest nonempty class and replaces it with the new page. The referenced bit will be cleared periodically to distinguish the pages that are referenced recently from those that are not.

Working:

Reference String: 7 0 1 2 0 1 #1 3 2 #2 0 4 2 3 0 3 2



Page Reference	7	0	1	2	0	1	#1	3	2	#2	0	4	2	3	0	3	2
Frame 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	7 0	7 0	7 0	7 0	7 0	7 0	7 0	3 0	3 0	3 0	3 0	4 0	4 0	3 0	3 0	3 0	3 0
Frame 2		0	0	0	1	1	1	1	0	0	1	1	0	0	1	1	1
		0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0
Frame 3			0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
			1 0	1 0	1 0	1 0	1 1	1 1	1 1	1 1	1 1	1 1	1 1	1 1	1 1	1 1	1 1
Frame 4				0	0	0	0	0	1	1	1	1	1	1	1	1	1
				2 0	2 0	2 0	2 0	2 0	2 0	2 1	2 1	2 1	2 1	2 1	2 1	2 1	2 1
Misses	1	2	3	4	4	4	4	5	5	5	5	6	6	7	7	7	7

Algorithm	Frame count	Ref count	Page faults	Hits	Hit ratio	Miss Ratio
NRU	4	15	7	8	0.533333	0.466667

Implementation:

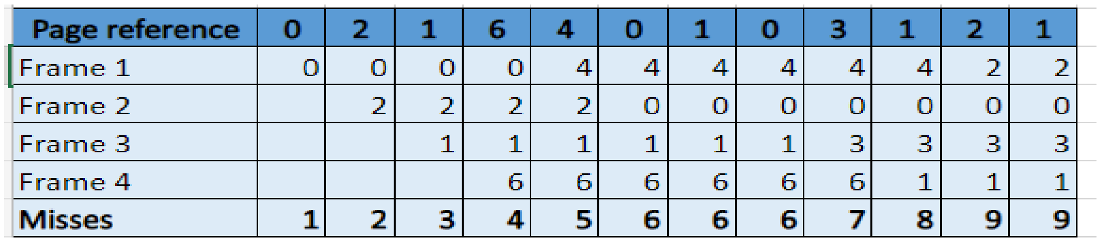
To maintain the classes a map is taken and whenever a page is referenced the reference bit is updated and when modified the modified bit is updated. The class of the page is changed.

To indicate the modification # symbol is used in the input.

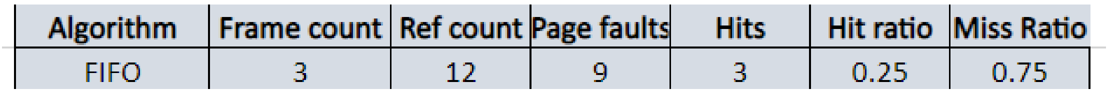
4. **First-In-First-Out (FIFO)**

This algorithm works on the principle of "First in First Out". In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal. Balady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. FIFO page replacement algorithm is definitely not the best page replacement algorithm to use practically. When the number of incoming pages is less, and a user is looking for a simple approach, FIFO might be a reasonable choice.

Working



Page reference	0	2	1	6	4	0	1	0	3	1	2	1
Frame 1	0	0	0	0	4	4	4	4	4	4	2	2
Frame 2		2	2	2	2	0	0	0	0	0	0	0
Frame 3			1	1	1	1	1	1	3	3	3	3
Frame 4				6	6	6	6	6	6	1	1	1
Misses	**1**	**2**	**3**	**4**	**5**	**6**	**6**	**6**	**7**	**8**	**9**	**9**



Algorithm	Frame count	Ref count	Page faults	Hits	Hit ratio	Miss Ratio
FIFO	3	12	9	3	0.25	0.75

5. **FIFO with second chance**

It is a modified version of FIFO algorithm. FIFO has a disadvantage that even a page is frequently and recently used, FIFO replaces this page because it came early. To overcome this disadvantage we use FIFO with second chance. We use reference bit that identifies the page is recently used or not. If reference bit is 0 , then the page is old, if reference bit is 1 then it is recently used. We don't replace the page with referenced bit equal to 1, we give this page a second chance without replacing immediately. If we visit a page with reference bit equal to 1 , then we clear the bit and move to next page. If reference bit is 0 then the page is replaced immediately.

Working

Page reference	0	4	1	4	2	4	3	4	2	4	0	4
Frame 1	0	0	0	0	2	2	2	2	2	2	2	2
Frame 2		4	4	4	4	4	4	4	4	4	4	4
Frame 3			1	1	1	1	3	3	3	3	0	0
Misses	**1**	**2**	**3**	**3**	**4**	**4**	**5**	**5**	**5**	**5**	**6**	**6**

Algorithm	Frame count	Ref count	Page faults	Hits	Hit ratio	Miss Ratio
FIFO SC	3	12	6	6	0.5	0.5

6. Clock

The clock algorithm keeps a circular list of pages in memory, with the "hand" (iterator) pointing to the last examined page frame in the list. When a page fault occurs and no empty frames exist, then the R (referenced) bit is inspected at the hand's location. If R is 0, the new page is put in place of the age the "hand" points to, and the hand is advanced in one position. Otherwise, the R bit is cleared, then the clock hand is incremented, and the process is repeated until a page is replaced.

Working:

Page reference: 2,1,5,2,4,5,3,2,5,3
Number of frames: 3

Memory trace:

Page reference	2	1	5	2	4	5	3	2	5	3
Frame 1	2*	2*	2*	2*	4*	4*	4*	4	5*	5*
Frame 2		1*	1*	1*	1	1	3*	3	3	3*
Frame 3			5*	5*	5	5*	5*	2*	2*	2*
Misses	**1**	**2**	**3**		**4**		**5**	**6**	**7**	
Pointer	1	2	0	0	1	1	2	0	1	1

Algorithm	Frame count	Ref count	Page faults	Hits	Hit ratio	Miss Ratio
Clock	3	10	7	3	0.3	0.7

Note: * means reference bit is set to 1 and pointer points to the next frame from where we start unsetting the reference bit until a frame found which has reference bit value is zero.

Implementation:

In the C++ implementation of clock page replacement algorithm, the main memory is simulated using a vector of length equal to number of frames. Reference string is considered as vector of numbers. Initially all frames are empty and this scenario is indicated by inserting –1 into frames. Next, each page is accessed and while accessing the age we will get hit or miss and if it is hit, we simply access the next page but if it is miss then we will check for an empty fame first and if we found empty frame then that frame would be replaced by that page or else, we will find frame according to clock algorithm and that will insert current page into that frame.

A structure is being created to maintain all details about reference bit, page number, frame number. When we insert the page for into a frame then the old structure of that frame will be replaced by this new structure. When we get a page fault then we will search for unset frame in that structure and we will replace that frame.

struct Clock{

long long page_num, frame_num, clock_bit;

}

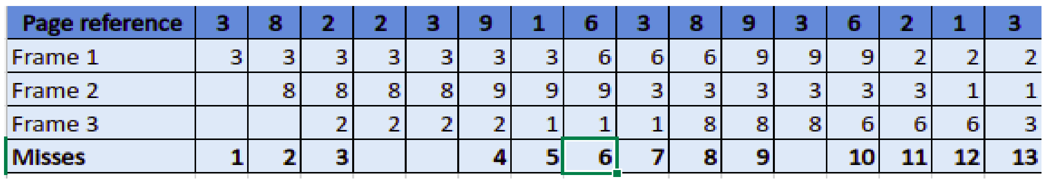
7. Least Recently Used (LRU)

LRU follows temporal locality which says an instruction that is recently executed has high chances of execution again. So, in this algorithm, we will not replace those pages which are recently used, and we will replace pages that are least recently used (pages that have not been referred to as the longest period). It is used often, and it is a good algorithm. Whenever there is the unavailability of a newly referred page in memory, page fault occurs and in response to that page fault operating system removes the least recently used page and that frame will be replaced by newly referred page.

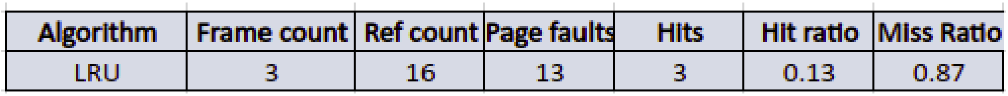
Working:

Page references: 3,8,2,3,9,1,6,3,8,9,3,6,2,1,3
Number of frames: 3

Memory trace:



Page reference	3	8	2	2	3	9	1	6	3	8	9	3	6	2	1	3
Frame 1	3	3	3	3	3	3	3	6	6	6	9	9	9	2	2	2
Frame 2		8	8	8	8	9	9	9	3	3	3	3	3	3	1	1
Frame 3			2	2	2	2	1	1	1	8	8	8	6	6	6	3
Misses	**1**	**2**	**3**			**4**	**5**	**6**	**7**	**8**	**9**		**10**	**11**	**12**	**13**



Algorithm	Frame count	Ref count	Page faults	Hits	Hit ratio	Miss Ratio
LRU	3	16	13	3	0.13	0.87

Implementacion:

In the C++ implementation of clock page replacement algorithm, the main memory is simulated using a vector of length equal to number of frames. Reference string is considered as vector of numbers. Initially all frames are empty and this scenario is indicated by inserting –1 into frames.

Next, each page is accessed and while accessing the age we will get hit or miss and if it is hit, we simply access the next page but if it is miss then we will check for an empty fame first and if we found empty frame then that frame would be replaced by that page or else, we will find frame according to LRU algorithm and that will insert current page into that frame.

A vector holds the reference order of pages and whenever there is a page fault first item in that vector will be considered for replacement and the newly accessed page will be pushed back to the vector.

8. **Not Frequently Used (NFU):**

Not Frequently used algorithm also known as Least Frequently used algorithm. In this algorithm system keep track of number of times block is referred to in memory. When a page is to be replaced it will be replaced with the block having least frequency. If more than one page has the same frequency, then it will follow First-In-First-Out replacement algorithm. The main problem in NFU is that it keeps track of frequency of use rather than time span of usage of block.

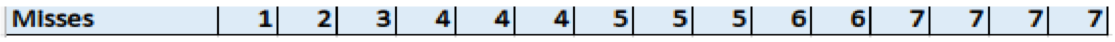
Working:

Reference string: 7 0 1 2 0 1 #1 3 2 #2 0 4 2 3 0 3 2

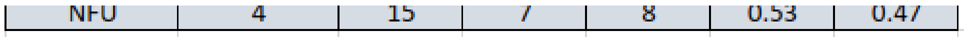
- # Represents that a page is being modified. As this algorithm does not use modified bit, these will be ignored.

Number of page frames: 4

Below is the memory trace:



Misses	1	2	3	4	4	4	5	5	5	6	6	7	7	7	7



NFU	4	15	7	8	0.53	0.47

Implementation:

Used unordered map and vector to implement NFU algorithm. Every time update the frequency in map if key Is present. If not present, then add the frame with frequency 1. If frames are full then remove the frame with low frequency and make the frequency as zero. If more than one frame has the same frequency, then follow FIFO.

9. **Working Set:**

Working set algorithm is best solution for Thrashing. The key idea is to remove pages even there is no page fault. The no of frames allocated over time varies. If the working window size is n it will keep only last n referred pages in memory. Other pages will be removed even when No page fault occurs. Implementation of the working set will perform implicit load control.

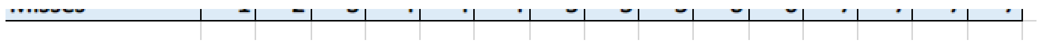
Working:

Reference string: 7 0 1 2 0 1 #1 3 2 #2 0 4 2 3 0 3 2

- # Represents that a page is being modified. As this algorithm does not use modified bit, these will be ignored.

Number of page frames: 4

Below is the memory trace:



Misses	1	2	3	4	4	4	5	5	5	6	6	7	7	7	7

Summary:

working set	4	15	7	8	0.53	0.47

Implementation:

Used simple vector for implementation. For Every iteration check the first element in working set and remove if it should not be present in working window. If the given string is present in the set, then it is hit. If not present, then push the value into vector. Maintain the FIFO in case of collision.

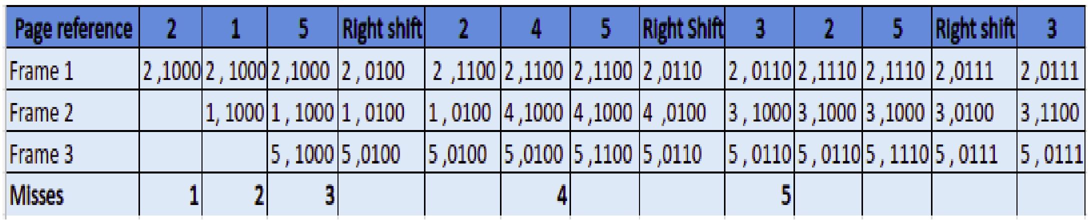
10. Aging (approximate LRU)

It uses a bit field of w bits for each page to track its accessing profile. Every time a page is read, the first (i.e., most significant) bit of the page's bit field is set. For every n instructions all bit fields are right shifted by one bit. The next page to replace is the one with the lowest (numerical) value of its bit field. If there are several pages having the same value, an arbitrary page is chosen. The aging algorithm works very well in many cases, and sometimes even better than LRU because it looks behind the last access. It furthermore is easy to implement because there are no expensive actions to perform when reading a page. However, finding the page with the lowest bit field value usually takes some time. Thus, it might be necessary to predetermine the next page to be swapped out in the background

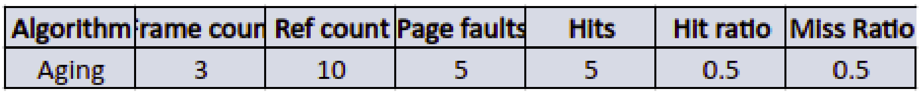
Working:

Page references : 2,1,5,2,4,5,3,2,5,3
Number of frames: 3

Memory trace:



Page reference	2	1	5	Right shift	2	4	5	Right Shift	3	2	5	Right shift	3
Frame 1	2 ,1000	2 , 1000	2 ,1000	2 , 0100	2 ,1100	2 ,1100	2 ,1100	2 ,0110	2 , 0110	2 ,1110	2 ,1110	2 ,0111	2 ,0111
Frame 2		1, 1000	1 , 1000	1 , 0100	1 , 0100	4 ,1000	4 ,1000	4 ,0100	3 , 1000	3 ,1000	3 ,1000	3 ,0100	3 ,1100
Frame 3			5 , 1000	5 ,0100	5 ,0100	5 ,0100	5 ,1100	5 ,0110	5 , 0110	5 , 0110	5 , 1110	5 , 0111	5 , 0111
Misses	1	2	3			4			5				



Algorithm	Frame count	Ref count	Page faults	Hits	Hit ratio	Miss Ratio
Aging	3	10	5	5	0.5	0.5

Implementation:

In the C++ implementation of clock page replacement algorithm, the main memory is simulated using a vector of length equal to number of frames. Reference string is considered as vector of numbers. Initially all frames are empty and this scenario is indicated by inserting –1 into frames.

Next, each page is accessed and while accessing the age we will get hit or miss and if it is hit, we simply access the next page but if it is miss then we will check for an empty fame first and if we found empty frame then that frame would be replaced by that page or else, we will find frame according to LRU algorithm and that will insert current page into that frame.

A structure is being created to maintain details about the binary number bits , page number, and frame number. When a page referred, we access that page bits and set the MSB bit. When there is a page fault then we sort this structure and will choose the least binary-valued page and will replace that.
Right rotation operation is being performed after every **x** page accesses where **x** is equal to number of frames.

```
struct aging{
long long frame_num, page_num;
        string bits=”00000000”; // 8 bits binary number
}
```

11. **WSClock**
This algorithm combines some of the best features of WS and CLOCK. In this algorithm we keep track of time and referenced bit. All pages are kept in a circular list. As pages are added, they go into the ring. The “clock hand” advances around the ring. Each entry contains “time of last use”. WSClock walks through the frames in order, looking for a good candidate for replacement, cleaning the reference bits as it goes. If the frame has been referenced since it was last inspected, it is given a second chance. We place frames in a circle with single clock hand. Due to its simplicity of implementation and good performance, it is widely used in practice.

Title, Introduction, Problem Description, Solution Approach, , , , Result, and conclusion

Work Distribution

The work done by each member of the team is as follows:

- Anjaneyulu Bairi 2021202008
  - LRU
  - Aging(approximation of LRU)
  - Clock
  - Python graph
  - Report

- Praveen Bandaru 2021202012
  - FIFO
  - FIFO Second Chance
  - WSClock
  - Report
- Sai Shushma Maturi 2021201012
  - Random
  - Not Frequently Used
  - Working Set
  - Report
- Sowmya Lahari Vajrala 2021202010
  - Optimal Algorithm
  - Not Recently Used
  - Driver code
  - Report
  - Python graph

Problems Faced and Shortcomings:

- For Aging algorithm and WSClock algorithm the resources were less and suggested different approaches
- For NRU instead of using time to clear reference bit, a counter is used.

Learnings:

- Clear idea on how to implement various algorithms.
- Performance of each algorithm

RESULTS:

Inputs:

Test Case 1:

Frames: 3

References: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

Test Case 2:

Frames: 4

References: 1,2,1,#1,3,#3

Test case 3:

Frames: 4

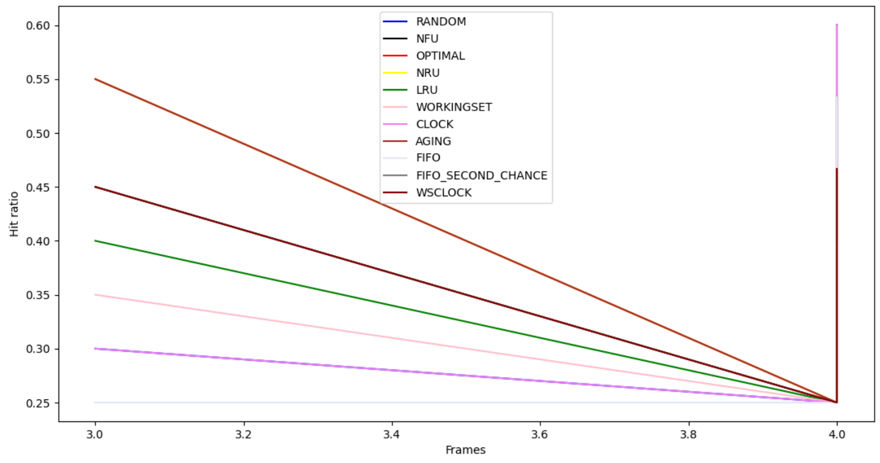
References: 7,0,1,2,0,1,#1,3,2,#2,0,4,2,3,0,3,2

Test case 4:

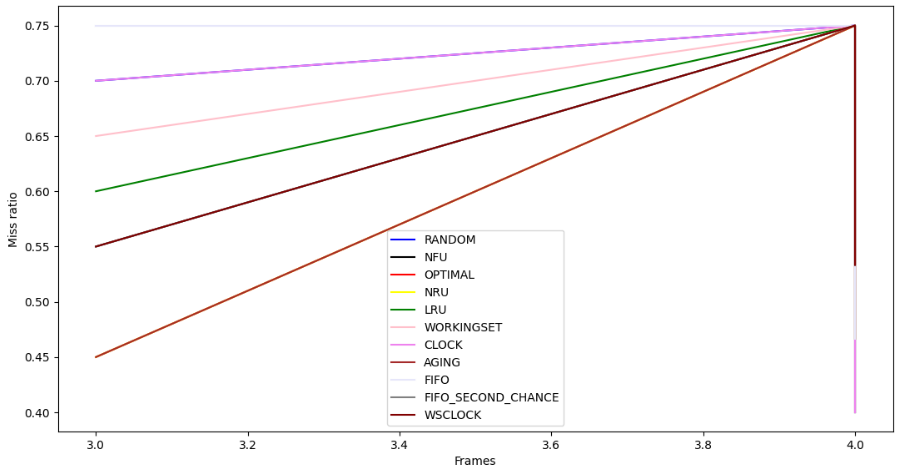
Frames: 4

Refernces: 7,0,1,2,0,1,3,2,0,4,2,3,0,3,2

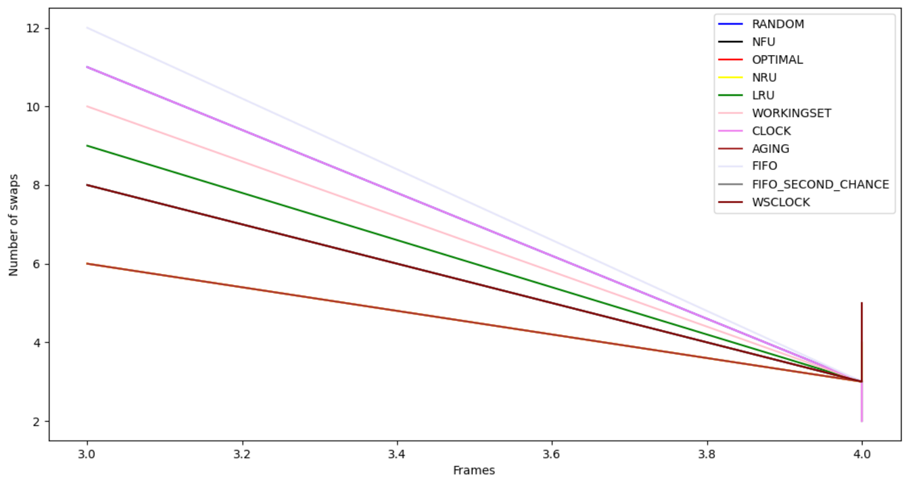
Graph between number of frames and hit ratio:



Graph between number of frames and miss ratio:



Graph between number of frames and number of page swaps:



CONCLUSION:

The evolution of replacement algorithms shows the analyses and proof of better performance has moved from mathematical analysis to testing against real world program traces. This trend shows how difficult it is to mathematically model the memory behavior of programs.