PAGE REPLACEMENT POLICIES SIMULATION

Team Name: Team CDSP
Project Title:Page Replacement Policies Simulation.
Mentor:Mahesh.
Team Members:

- Chegu sai poorna chandu (2022201062)
- Nemalikanti V M Dheeraj (2022201022)
- Patha Sai Sumith (2022202004)
- Kanamarlapudi Prem sai kumar (2022201036)

Introduction to Page Replacement Algorithms:

Page replacement is needed in the operating systems that use virtual memory using Demand Paging. As we know that in Demand paging, only a set of pages of a process is loaded into the memory. This is done so that we can have more processes in the memory at the same time.

When a page that is residing in virtual memory is requested by a process for its execution, the Operating System needs to decide which page will be replaced by this requested page. This process is known as page replacement and is a vital component in virtual memory management.

Use Of Page Replacement Algorithms in Operating System:

To understand why we need page replacement algorithms, we first need to know about page faults. Let's see what is a page fault.

Page Fault: A Page Fault occurs when a program running in CPU tries to access a page that is in the address space of that program, but the requested page is currently not loaded into the main physical memory, the RAM of the system.

Since the actual RAM is much less than the virtual memory the 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.**

Algorithms:

1. Random Page Replacement Algorithm

Random replacement algorithm replaces a random page in memory. This eliminates the overhead cost of tracking page references. Usually, it fares better than FIFO, and for looping memory references it is better than LRU, although generally, LRU performs better in practice.

page reference	0	1	2	6	4	0	1	0	3	1	2	1
frame1	0	0	0	0	0	0	0	0	0	0	0	0
frame2		2	2	2	1	1	1	1	1	1	1	1
frame3			1	1	6	6	6	6	4	4	3	3
frame4				6	4	4	4	4	3	3	2	2
misses	1	2	3	4	5	5	5	5	6	6	7	7

Algorithm	frames	refs	hits	misses	hit ratio	misss ratio
random	4	12	5	7	0.41	0.58

Implementation:

1. Used vector to store pages
2. Generated a random index using rand() function and deleted the page from the vector.
3. If the page is present in the vector, then it's a hit else it's a miss

Pros:

1. Easy to implement
2. Often gives decent results

Cons:

1. Sometimes there are more page faults

2) Optimal Page Replacement Algorithm

Optimal page replacement is the best page replacement algorithm as this algorithm results in the least number of page faults. In this algorithm, the pages are replaced with the ones that will not be used for the longest duration of time in the future. In simple terms, the pages that will be referred farthest in the future are replaced in this algorithm.

page sequence	7	0	1	2	0	3	0	4	2	3	0	3	2	1
Frame1	7	0	1	2	2	3	3	4	4	4	0	0	0	1
Frame2		7	0	1	1	2	2	3	3	3	3	3	3	0
Frame3			7	0	0	0	0	2	2	2	2	2	2	3
Misses	1	2	3	4	4	5	5	6	6	6	7	7	7	8

Algorithm	frames	refs	hits	misses	hit ratio	miss ratio
OPTIMAL	3	14	6	8	0.428	0.571

Implementation:

1. Page sequence is taken in a vector and the frames that are present in the system are taken in an unordered_set.
2. Iterate over the page sequence
    a. If number of pages in the system are less than frame size then insert the page to the set.
    b. If number of pages in the system is equal to the frames size then delete a page that will not be used in nearest future and insert the new page .

PROS:

1. Complexity is less and easy to implement.
2. Simple datastructures can be used.

CONS: Not possible in practical situations because future page requests are not known.

3)NRU

The not recently used (NRU) page replacement algorithm is an algorithm that favours keeping pages in memory that have been recently used. This algorithm works on the following principle: when a page is referenced, a referenced bit is set for that page, marking it as referenced. Similarly, when a page is modified (written to), a modified bit is set.

At a certain fixed time interval, a timer interrupt triggers and clears the referenced bit of all the pages, so only pages referenced within the current timer interval are marked with a referenced bit. When a page needs to be replaced, the operating system divides the pages into four classes:

3. referenced, modified
2. referenced, not modified
1. not referenced, modified
0. not referenced, not modified

Although it does not seem possible for a page to be modified yet not referenced, this happens when a class 3 page has its referenced bit cleared by the timer interrupt. The NRU algorithm picks a random page from the lowest category for removal. So out of the above four page categories, the NRU algorithm will replace a not-referenced, not-modified page if such a page exists. Note that this algorithm implies that a modified but not-referenced (within the last timer interval) page is less important than a not-modified page that is intensely referenced.

Note: # means modified page

page Sequence	7	0	1	2	0	1	#1	3	2	#2	0	4	2	3	0	3	2
class 0					7,2	7	7		0	0			0,4	0			3,0
class 1									1	1	1						
class 2	7	7,0	7,0,1	7,0,2	0	0,1	0	0,3	2		0	0,4		3	3,0	3,0	
class 3							1	1		2	2	2	2	2	2	2	2

Algorithm	frames	refs	hits	misses	hit ratio	miss ratio
NRU	3	17	8	9	0.47	0.52

Implementation:

1. Take a map where key specifies the class the of the page and values contains the pages that belongs to the class.
    - 0-not referenced,not modified
    - 1-not referenced,modified
    - 2-referenced,not modified
    - 3-referenced,modified
2. Consider a variable clock_counter which will remove all the reference bits for every ith iteration where i%clock_counter==0

    If the above condition is true then shift all the pages in class 2,3 to class 0,1 respectively.

3. If the page is modified page then it should be compulsory in the main memory and change its class to 1,3 from class 0,2 respectively.
4. If the pages in the system are equal to the frame size then remove a page randomly from the lowest class(0-lowest,3-highest) and insert the page in class 2.

PROS:

- It is implementable

CONS:

- Require scanning through reference and modified bits.

4. FIFO Page Replacement Algorithm

This is the simplest page replacement algorithm. In this algorithm, operating system keeps track of all pages in the memory in a queue, 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. Belady'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.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	page reference	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2
2	frame 1	7	7	7	0	0	1	2	3	0	4	2	2	2	3	0
3	frame 2		0	0	1	1	2	3	0	4	2	3	3	3	0	1
4	frame 3			1	2	2	3	0	4	2	3	0	0	0	1	2
5	misses	1	2	3	4	4	5	6	7	8	9	10	10	10	11	12
6																
7	Algorithm	frames	refs	hits	misses	hit ratio	misss ratio									
8	fifo	3	15	3	12	0.2	0.8									

Implementation:

1. Implemented using a queue data structure
2. If page is present in queue, it's a hit.
3. If the queue is full, remove the front element of the queue and push the current page of the queue

PROS:

1. It is simple and easy to understand & implement
2. It is efficiently used for small systems

CONS:

1. The process effectiveness is low.
2. When we increase the number of frames while using FIFO, we are giving more memory to processes. So, page faults should decrease, but here the page faults are increasing. This problem is called belady's anomaly.

5) FIFO with Second Chance

A modified form of the FIFO page replacement algorithm, known as the Second-chance page replacement algorithm, fares relatively better than FIFO at little cost for the improvement. It works by looking at the front of the queue as FIFO does, but instead of immediately paging out that page, it checks to see if its referenced bit is set. If it is not set, the page is swapped out. Otherwise, the referenced bit is cleared, and this process is repeated. This can also be thought of as a circular queue. If all the pages have their referenced bit set, on the second encounter of the first page in the list, that page will be swapped out, as it now has its referenced bit cleared. If all the pages have their reference bit cleared, then second chance algorithm degenerates into pure FIFO.

Implementation:

1.This algorithm is implemented using double ended queue and unordered map.

2.When the page present in frame is referenced again then its bit is set to 1 using unordered map.

3.Thus if frame is full and we want to replace it we will give the page second chance by resetting its reference bit.

Page Sequence	7	0	1	2	0	1	1	3	2	2	0	4	2	3	0	3	2
Frame1	7	7	7	2	2	2	2	3	3	3	3	4	4	4	4	4	2
Frame2		0	0	0	0	0	0	0	2	2	2	2	2	2	0	0	0
Frame3			1	1	1	1	1	1	1	1	0	0	0	3	3	3	3
Misses	1	2	3	4	4	4	4	5	6	6	7	8	8	9	10	10	11

Algorithm	Frames	Refs	Hits	Misses	Hit Ratio	Miss Ratio
FIFO With Second Chance	3	17	6	11	0.35	0.647

PROS:

1.It is simple and easy to understand and implement.

2.It is efficiently used for small systems.

CONS:

1.Even though it is modified version of FIFO there is chance of Belady's Anomaly.

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.

Implementation:

Create an array frames to track the pages currently in memory and another Boolean array second_chance to track whether that page has been accessed since it's last replacement (that is if it deserves a second chance or not) and a variable pointer to track the target for replacement.

1.Start traversing the array arr. If the page already exists, simply set its corresponding element in second_chance to true and return.

2.If the page doesn't exist, check whether the space pointed to by pointer is empty (indicating cache isn't full yet) – if so, we will put the element there and return, else we'll traverse the array arr one by one (cyclically using the value of pointer), marking all corresponding second_chance elements as false, till we find a one that's already false. That is the most suitable page for replacement, so we do so and return.

3.Finally, we report the page fault count.

CONS:

1.Clock may suffer from Belady's Anomaly.

Page Sequence	7	0	1	2	0	1	1	3	2	2	0	4	2	3	0	3	2
Frame1	7	7	7	2	2	2	2	3	3	3	3	4	4	4	0	0	0
Frame2		0	0	0	0	0	0	0	2	2	2	2	2	2	2	2	2
Frame3			1	1	1	1	1	1	1	1	0	0	0	3	3	3	3
Misses	1	2	3	4	4	4	4	5	6	6	7	8	8	9	10	10	10

Algorithm	Frames	Refs	Hits	Misses	Hit Ratio	Miss Ratio
FIFO With Second Chance	3	17	7	10	0.41	0.588

7. LRU Page Replacement Algorithm

LRU stands for Least Recently Used. As the name suggests, this algorithm is based on the strategy that whenever a page fault occurs, the least recently used page will be replaced with a new page. So, the page not utilized for the longest time in the memory (compared to all other pages) gets replaced

page reference	0	1	2	6	4	0	1	0	3	1	2	1
frame1	0	2	1	6	4	0	1	0	3	1	2	1
frame2		0	2	1	6	4	0	1	0	3	1	2
frame3			0	2	1	6	4	4	1	0	3	3
frame4				0	2	1	6	6	4	4	0	0
misses	1	2	3	4	5	6	6	6	7	7	8	8

Algorithm	frames	refs	hits	misses	hit ratio	misss ratio
LRU	4	12	4	8	0.33	0.66

Implementation:

1. Consider a doubly linked list where head contains the most recently used page And tail contains least recently used page.
2. To find whether a page is already present in Frames considered an unordered_map where key is the page number and value is a pointer to the node in the doubly linked list
3. Iterate over the page sequence:
    a. If the page is present in the system then make that page as head of the list
    b. If the number of pages in system is less than frame size then insert the page at the head of the list
    c. If the number of pages in system is equal to frame size then delete the page present in the tail and insert the required page at the head of the list.

PROS:

1. The page in the main memory that hasn't been used in the longest will be chosen for replacement.
2. It gives fewer page faults than any other page replacement algorithm. So, LRU is the most commonly utilized method.
3. It is a very effective algorithm.
4. LRU doesn't suffer from Belady's Anomaly.

CONS:

1. It is expensive and more complex.

2. It needs an additional data structure.
3. It isn't to implement because it requires hardware assistance.

8. NFU Page Replacement Algorithm

Whenever a new page is referred to and is not present in memory, the page fault occurs and the Operating System replaces one of the existing pages with a newly needed page. NFU is one such page replacement policy in which the least frequently used pages are replaced. If the frequency of pages is the same, then the page that has arrived first is replaced first.

page reference	5	0	1	3	2	4	1	0	5
frame1	5	0	1	3	2	4	1	0	5
frame2		5	0	1	3	2	4	1	0
frame3			5	0	1	3	2	4	1
frame4				5	0	1	3	2	4
misses	1	2	3	4	5	6	6	7	8
Algorithm	frames	refs	hits	misses	hit ratio	misss ratio			
NFU	4	9	1	8	0.11	0.88			

Implementation:

1. Consider a map where key is the page sequence and value is an LRU list.
2. Iterate over all the pages
    a. If the page is already in the system increase the frequence the frequency of the page and delete it from the previous frequency list and insert it into the new frequency list
    b. If the number of pages in system is less than frame size then insert the page in the frequency list of 1.
    c. If number of pages in system is equal to frame size then delete a Least recently used page from the un empty least frequency list and insert the required page in the frequency list of 1.

PROS:

- Implementable

9. Working Set Replacement Algorithm

The working set is a dynamic subset of a process's address space that must be loaded in main memory to ensure acceptable processing efficiency. Working set policies can be tuned for close-to- optimal throughput and response time. They prevent thrashing

page reference	0	1	2	6	4	0	1	0	3	1	2	1
frame1	0	0	0	0	1	0	0	0	0	0	0	1
frame2		2	1	1	2	1	1	1	1	1	1	2
frame3			2	3	4	4	4	4	3	3	2	3
frame4				6	6	6	6				3	
misses	1	2	3	4	5	6	7	7	8	8	9	9

Algorithm	frames	refs	hits	misses	hit ratio	misss ratio
working set	4	12	3	9	0.25	0.75

Implementation:

1. Used a set and deque to implement
2. Used sliding window technique
3. I pushed the pages into a deque and then converted it to a set
4. For every iteration check if the page is present in the set or not. If it's present, then it's a hit else miss.
5.

Pros:

1. In other words, the working set strategy prevents thrashing while keeping the degree of multiprogramming as high as possible.
2. It optimizes CPU utilization and throughput.

Cons

1. If the allocated frames are lesser than the size of the current locality, the process is bound to thrash

10.Aging Algorithm

We have a bit field of w bits for each page in aging algorithm.After every 3 page references we right shift bit field of every page reference. If the referenced page is already present in the frame we make the left most bit of bit field of that frame to 1. When the page is first referenced only the left most bit of the bit field of the frame will be 1.

page references	5	0	1	3	2	4	1	0	5
frame1	5	0	1	3	3	3	3	0	5
frame2		5	0	1	1	1	1	1	1
frame3			5	0	0	4	4	4	4
frame4				5	2	2	2	2	2
missed	1	2	3	4	5	6	6	7	8

Algorithm	frames	refs	hits	misses	hit-rato	miss-ratio
Working Set	4	9	1	8	0.11	0.88

Implementation:

Create a vector free to keep track of the pages in the memory.Create another vector cache of type pages in which each node stores page number,frame number and counter.We use the utility function add_page to check if the page is present in the cache,if not then if there is an empty frame present in the cache it will add the current reference.It will return 0 if the page is not present in the cache and the cache is full.In that case we will sort the vector and replace the first element with the current reference and the first bit of the counter is set to 1.
Right rotation of counter of all pages is made after every 3 successful page references

Pros:

1. Aging can offer near-optimal performance for a moderate price.
2.Aging ensures that pages referenced more recently, though less frequently referenced, will have higher priority over pages more frequently referenced in the past.

11.WS Clock

The basic working set algorithm is cumbersome since the entire page table has to be scanned at each page fault until a suitable candidate is located. An improved algorithm, that is based on the clock algorithm but also uses the working set information is called WSClock (Carr and Hennessey, 1981). Due to its simplicity of implementation and good performance, it is widely used in practice.

The data structure needed is a circular list of page frames.. Initially, this list is empty. When the first page is loaded, it is added to the list. As more pages are added, they go into the list to form a ring. Each entry contains the *Time of last use* field from the basic working set algorithm, as well as the *R* bit (shown) and the *M* bit (not shown).

As with the clock algorithm, at each page fault the page pointed to by the hand is examined first. If the *R* bit is set to 1, the page has been used during the current tick so it is not an ideal candidate to remove. The *R* bit is then set to 0, the hand advanced to the next page, and the algorithm repeated for that page.

page references	5	0	1	3	2	4	1	0	5
frame1	5	0	1	3	3	3	3	0	5
frame2		5	0	1	1	1	1	1	1
frame3			5	0	0	4	4	4	4
frame4				5	2	2	2	2	2
missed	1	2	3	4	5	6	6	7	8
Algorithm	frames	refs	hits	misses	hit-rato	miss-ratio			
Working Set	4	9	1	8	0.11	0.88			

Implementation:

1.Create two unordered maps one maps the page reference to the frame no,the second map maps the frame no to the details of the frame.

2.Using for loop i will iterate through each page reference.At each page reference i will check if the page is already present in the cache or not,if not then i will check if there are any empty frames using the total variable which contains the number of occupied frames.If there are any empty frames then i will insert the current page reference.

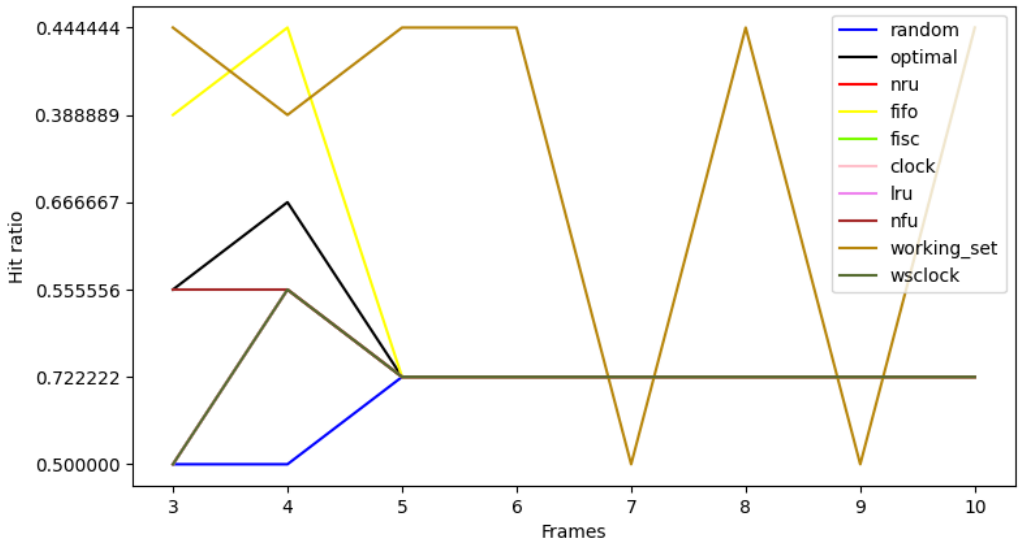
3.We have a clock pointer pointing to the first element inserted.If the first element flag bit is 1 we make it 0 and move to the next element else if it's flag bit is 0 we check the threshold .If it satisfies the threshold then replace the page.

PROS:

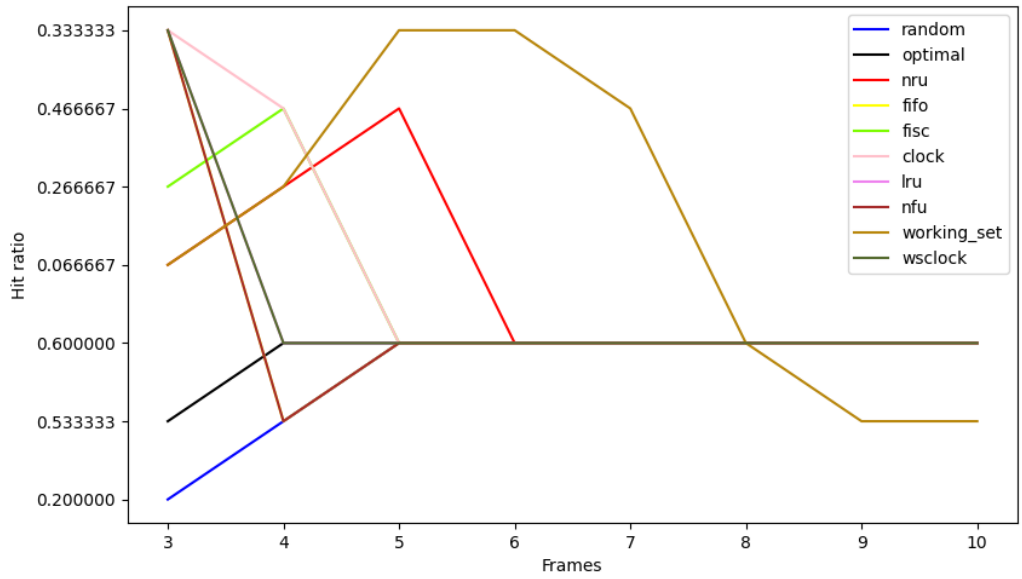
The new algorithm combines the most useful feature of WS-a natural and effective load control that prevents thrashing-with the simplicity and efficiency of CLOCK.
WSClock performs as effective as WS without OS overhead.

Graphs for hit ratio:

Input Sequence: 0 4 1 4 2 4 3 4 2 4 0 4 1 4 2 4 3 4
Frame size :3



Input Sequence: 7 0 1 2 0 1 3 2 0 4 2 3 0 3 2
Frame size :3



Work Distribution:

- chandu
    - LFU
    - NRU
    - Optimal
    - LRU
    - report
- Dheeraj
    - FIFO
    - Working set
    - Random
    - Graphs
    - report
- Sumith
    - FIFO with second chance
    - Clock
    - Power point presentation
    - report
- Prem
    - Aging
    - Wsclock
    - report