IIIT Hyderabad

Advanced Operating Systems

CS3.304 [M21]

Page Replacement Algorithms Simulations

Authors:
Devansh Avasthi
Kilaru Yasaswi Sri Chandra Gandhi
Kaustuv Dash
Rohit Jangir

Course Instructor:
Dr. Manish Shrivastava

November 27, 2021

Contents

0 General Details

Team Number: 2

Team Name: `team`

Team Members:

- Devansh Avasthi [2021201027]
- Kilaru Yasaswi Sri Chandra Gandhi [2021201080]
- Rohit Jangir [2021202013]
- Kaustuv Dash [2021202019]

Abstract

The aim of this project is to simulate various page replacement algorithms and measure their performances against an input page sequence for a given frame size. Another aim is to find out how the hit-ratio is affected by the increase in frame sizes. Lastly, we also aim to understand whether the number of swaps for a given page sequence is affected by the frame size. These comparisons are made across multiple page replacement algorithms.

1 Problem Statement

Write a memory management simulator that uses paging and measure the performances of page replacement algorithms.

The simulator should be able to read a memory trace, keep track of the loaded pages, check if an incoming page is already loaded, and replace a page if it is not.

The page sequence is read from a file and the user should be able to select the number of frames in memory.

The following page replacement algorithms need to be simulated:

- Random
- Optimal
- Not Recently Used

- First-In First-Out
- First-In First-Out with 2nd Chance
- Clock
- Least Recently Used
- Not Frequently Used
- Working Set
- Aging
- Working Set Clock

1.1 Output

On the console, the simulator should print the page sequence, and the hit ratio.

The page sequence will highlight the pages that are loaded or found and the pages that are replaced.

Graph showing performances of various algorithms. Hit-rate vs. number of frames.

Another graph based on user choice that shows hit-rate vs. all the algorithms.

2 Implementation

2.1 Random Page Replacement Algorithm

The Random Page Replacement Algorithm works by selecting a random index in the set of frames and substituting the incoming page from the page sequence in it's place. An advantage of this algorithm is that while being straightforward, it removes the need of tracking page references. It is a good baseline for comparison against more sophisticated algorithms.

A `randomIndex` is generated by obtaining the modulo with respect to the frame size with the `rand()` function. The value at this index is erased and replaced with the current index in the page sequence. The time complexity is $O(N)$ where N is the size of the input sequence.

2.2 Optimal Page Replacement Algorithm

The theoretically optimal page replacement algorithm replaces the page that will be used farthest in the future. While we cannot implement this because of our inability to predict the future, we can use this as the best case algorithm for comparison.

2.3 Not Recently Used Page Replacement Algorithm

The Not Recently Used page replacement algorithm replaces the page that has not been used for the longest time and prefers to retain pages that have been used recently. A reference bit is set for every page when it is referenced and there is a modified bit that represents whether the page has been modified or not. During replacement, the algorithm prioritizes the pages in the order of referenced and modified, referenced and not modified, not referenced and modified, not referenced and not modified.

2.4 First-In First-Out Page Replacement Algorithm

This algorithm is quite straightforward to implement and understand. We use a queue to keep track of all pages in memory, everytime a replacement is needed, the algorithm selects the first page in the queue and removes it. There is no additional tracking required for this algorithm but it isn't very ideal because we remove page frames that have been longest in memory.

2.5 First-In First-Out with 2nd Chance Page Replacement Algorithm

This algorithm is similar to the First-In First-Out algorithm except that we give every frame a second chance by setting it's reference bit to 1 on reference. It gives better performance with not a lot of additional tracking. The algorithm lets pages stay in frame for one additional hit before they're replaced. An important boundary condition is when all the pages have an even number of hits, in this case, the algorithm reverts to vanilla FIFO.

We declare an array `chanceBit[2][frames]` that contains the pages in the frames as well as their reference bits. We only move to the next frame if there is no load in the current iteration.

2.6 Clock Page Replacement Algorithm

The clock algorithm is a more efficient version of the First-In First-Out with second chance algorithm. 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 page the "hand" points to, and the hand is advanced one position. Otherwise, the R bit is cleared, then the clock hand is incremented and the process is repeated until a page is replaced.

2.7 Least Recently Used Page Replacement Algorithm

LRU works on the idea that pages that have been most heavily used in the past few instructions are most likely to be used heavily in the next few instructions too. It chooses the page that was last referenced the longest time ago and assumes recent behavior is a good predictor of the near future. The major problem is that it requires updating for every page reference.

2.8 Not Frequently Used Page Replacement Algorithm

The not frequently used (NFU) page replacement algorithm requires a counter, and every page has one counter of its own which is initially set to 0. At each clock interval, all pages that have been referenced within that interval will have their counter incremented by 1. In effect, the counters keep track of how frequently a page has been used. Thus, the page with the lowest counter can be swapped out when necessary.

2.9 Working Set Page Replacement Algorithm

The working set algorithm only keeps the pages that a process has used in the last few time intervals. The major goal is to restrict the number of processes in the ready list so that all can have their working set of pages in memory. But this is very expensive in practice.

2.10 Aging Page Replacement Algorithm

The aging algorithm is a modification of the NFU algorithm that make it aware of the time span of use. It shifts the reference bits into counters and picks the page with the smallest counter to replace. It generally has a longer counter length that NFU but it is one of the most important practical algorithms.

2.11 Working Set Clock Page Replacement Algorithm

This algorithm is a combination of the clock and working set algorithms. It uses the clock algorithm to replace pages, but it also keeps track of the pages that are in the working set. When a page is swapped out, it is removed

from the working set. When a page is swapped in, it is added to the working set. The WSClock algorithm checks the page access time before removing a page from the frame memory. If the chance bit is set to 0, the algorithm checks whether the page age has crossed the threshold.

A threshold is used and initialized, `unsigned long tt = 100;` An array with 3 rows is declared to store the pages in the frame memory, the reference bits, and the age of the page. The `gettimeofday` function is used to update the time during every reference and also to check whether the age has crossed the threshold. If the age has crossed the threshold, the page is removed from the frame memory.

2.12 Driver Code

The main program offers a choice to the user regarding the simulation. The user can choose to simulate each algorithm individually, or all of them at once.

2.12.1 Single Algorithm Simulation

The user can choose to simulate a single algorithm. The user can choose to simulate the algorithm with a variable number of frames. The user can also choose to simulate the algorithm with a variable number of pages by changing the input in the `pages.txt` file.

Entering `0` will mean that the simulation ends and the graph that was generated with the hit-ratio vs. the frame size will be displayed to the user. Another option is to change directory to the `graphs` directory and then open the graph that was generated.

2.12.2 All Algorithms Simulation

The user can choose to simulate all the algorithms at once. The user can choose to simulate the algorithms with a variable number of frames. The user can also choose to simulate the algorithms with a variable number of pages by changing the input in the `pages.txt` file.

Similar to the above case, entering `0` will mean that the simulation ends and the graphs will be displayed, overlaid one on top of the other to the user. Another option is to change directory to the `graphs` directory and then open the graphs that were generated.

2.13 Printing Page Sequences

A function is implemented that prints the page sequences while highlighting the page hits in green and the page faults in red. This makes it easier to visualize the page sequence and the workings of the algorithm can be checked visually.

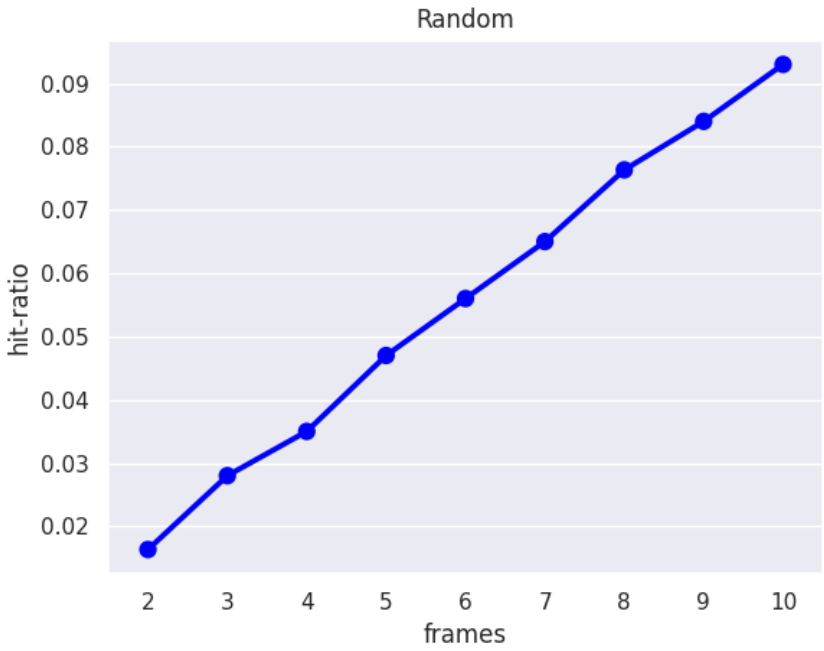
```
void printSet(unordered_set<int> &frameSet, int page, bool col)
{
    unordered_set<int>::iterator it;
    for (it = frameSet.begin(); it != frameSet.end(); it++)
        if (page == *it and col)
            cout << "\033[0;37m|"
                 << "\033[0;32m" << (*it);
        else if (page == *it)
            cout << "\033[0;37m|"
                 << "\033[0;31m" << (*it);
        else
            cout << "\033[0;37m|"
                 << "\033[0;37m" << (*it);
    cout << "\033[0;37m|";
}
```

2.14 Graphs

The graphs are printed by writing the output of each of the algorithms to a comma-separated value file. The graphs are then displayed to the user by using `seaborn`.

2.14.1 Hit-rate vs. Number of Frames

An example graph is shown below. The graph shows the hit-rate vs. the number of frames for the random algorithm. The graph is generated by using the `seaborn` command.




```
df = pd.read_csv("graph/random.csv")
img = sns.pointplot(x = 'frames', y = 'hit-ratio',data = df,color="blue")
```

3 Work Distribution

This project was completed with equal enthusiasm and contribution from all the team-members and specific tasks were assigned to each member.

- Driver Code - Devash Avasthi
- Random - Kilaru Yasaswi Sri Chandra Gandhi
- Optimal - Devash Avasthi
- NRU - Rohit Jangir
- FIFO - Kaustuv Dash
- FIFO 2nd chance - Kilaru Yasaswi Sri Chandra Gandhi

- Clock - Devash Avasthi
- LRU - Rohit Jangir
- NFU - Kaustuv Dash
- Working Set - Kaustuv Dash
- Aging - Rohit Jangir
- Working Set Clock - Kilaru Yasaswi Sri Chandra Gandhi
- Graphs - Devash Avasthi
- Final report - Kilaru Yasaswi Sri Chandra Gandhi

4 Problems Faced & Learnings

- Using the `rand()` function without seeding it with `NULL` always gives the same output and doesn't replicate a random algorithm. The solution was to use `srand` and seed the random number.
- The theory about the Not Recently Used Algorithms and the Aging Algorithm is not very clear. A lot of discussion with the team was needed to understand the algorithms better.
- Sending the outputs of the each simulation to a `csv` file was initially problematic because the contents in the files were being rewritten completely and a lot of data was lost. The solution was to open the file in append mode and then write the contents of the file.
- Calling a Python program from within a C++ program was achieved with the help of the `system()` function.

5 Shortcomings

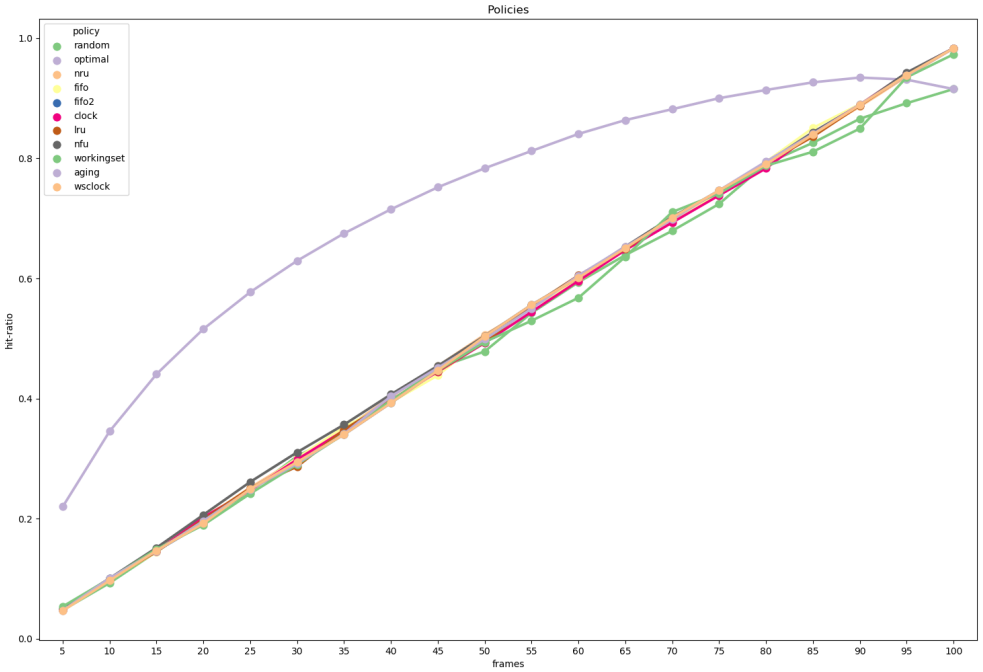
- The `system()` function is not portable. It is dependent on the operating system and the way it works is by suspending the program and calling the operating system to open the shell. It allocates the memory, runs the command and then frees the memory. This can be seen in the few moments that pass after ending a simulation.

- Running the simulations on large inputs was quite troublesome because all the algorithms performed imperceptibly close to one another. The best way to have a feel for the different algorithms is to either run the simulation on a small input or run them individually.
- A very untoward issue is the colour of the curves on the graph. The reason for this is that the `seaborn` library is not able to display more than 8 colours in it's palette.

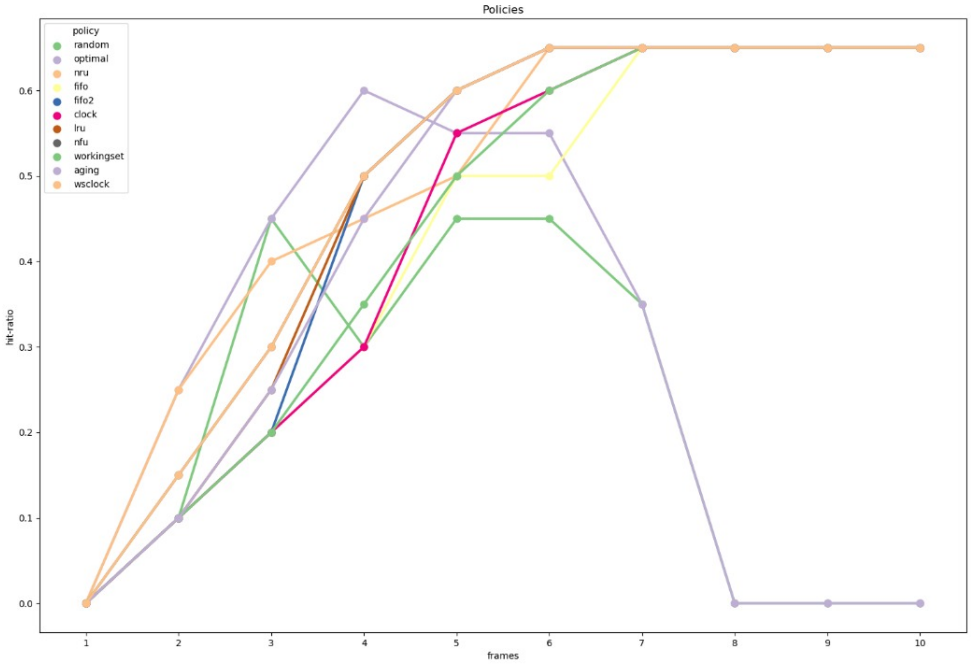
6 Results & Conclusion

On running the various simulations, we can confirm that our theoretical assumptions were correct. The optimal algorithm outperforms the other algorithms by a significant margin. The performance is most visible for frame sizes below 100. The WSClock algorithm takes time but for very large inputs, it is certainly close to the optimal algorithm.

Below is a graph with showing the performance of all the algorithms. Frame sizes upto a 100 have been simulated, most of the graphs overlap because of the nature of the input.



Here is a graph with showing the performance of all the algorithms on a smaller input and a much different page sequence. The Random algorithm performs poorly while the FIFO algorithm shows its shortcomings by not taking into account the number of times a page is referenced. The Second chance and Clock algorithms improve significantly but the Working Set Clock still matches the optimal algorithm.



7 Acknowledgements

We would like to thank the course instructor, Dr. Manish Srivastava and the teaching assistant in-charge of the project, Rishabh Malik for their help and guidance throughout the semester and in particular during the project.

We would like to thank the Linux Open Source Community, Python Open Source Community, and the C++ Open Source Community for their support on various forums and for the open-source software that is used in this project.

This report was typeset using LaTeX, originally developed by Leslie Lamport and based on Donald Knuth's TeX. The body text is set in 11 point Egenolff-Berner Garamond, a revival of Claude Garamont's humanist typeface.