Operating Systems Assignment 2

Semester 2 - 2024: Gerald Freislich A1853059

1. Introduction

When a process is being executed, on a modern computer, all instructions and required resources are loaded from the Hard Drive onto the RAM (Random Access Memory), so that the CPU can easily reach all requisite data (Aleks, 2020). Problematically, the RAM of a computer cannot hold all information from the Hard Drive, and hence, not all of the processes' required data can be loaded onto the RAM for the CPU. To solve this problem, Virtual Memory was created. Virtual Memory enables the CPU to feel that it has access to all data, by efficiently managing the transfer of required resources from the Hard Drive to RAM (Tyson J., 2000). When Virtual Memory is unable to manage memory efficiently; thrashing occurs. This is where the computer is constantly swapping resources from Hard Drive to RAM at the time that the resources are required by the CPU (Lenovo, nd). Such persistent swapping leads to poor computer performance and user experience.

For Virtual Memory to work well, pages and page tables were created. For every process that is executed a page table is created. This page table segments all the processes required resources into blocks, called pages, and points to their respective locations in memory (Grainger Illinois, 2021). During execution of the process, some pages of resources are loaded onto the RAM, whilst others may be left on the Hard Drive (Raza S., 2023). The role of Virtual Memory is to manage the pages loaded onto the RAM such that the CPU will have its requested data on the RAM at the right time. To do this, however, pages must be loaded off and onto RAM and hence some pages must be replaced. Multiple page replacement algorithms exist, and in this investigation three in specific are considered; these are Random, Least Recently Used (LRU) and Clock replacement. Random replacement randomly selects a page in the RAM to replace with the required page from the Hard Drive. LRU replacement, attaches a counter to each page, which monitors the last time a given page was used. The page on the RAM with the oldest time used is then replaced by the needed page from the Hard Drive. Clock replacement, however, attaches a reference or consideration bit to each page. It then cycles over all pages and finds the page that has not been accessed since its last consideration. Initially the reference or consideration bit for all pages is 1, and the Clock replacement algorithm changes these bits to zero, and then finds the first page where its reference bit is not 1 (meaning it was changed to zero during the last consideration traversal) (Geeks For Geeks, 2023). When a page is accessed this reference bit is changed to 1. When a page that the CPU requires is not on the RAM, a page fault occurs and it is following this page fault that a page replacement algorithm loads the required page onto the RAM, and replaces an existing page.

In this investigation, the page fault rate of each aforementioned page replacement algorithm is evaluated, for varying memory sizes on multiple applications. Memory sizes, here, denote

the number of frames of RAM that the computer has available. It is expected that as the number of frames increases, the number of page faults (or page fault rate) will decrease (CS University of Illinois Chicago, nd). The applications investigated include gcc, swim, sixtrack and bzip. There are three main questions to consider in this investigation, which are: "*How much memory does each traced program actually need?*", "*Which page replacement works best when having a low number of frames?*" and "*Does one algorithm work best in all situations?*". The report is structured as follows: in Section 2. We describe the methodology used for the subsequent experiments and experimental results provided in Section 3. In Section 4 we provide an analysis of results and in Section 5 we conclude the report.

2. Methodology

To evaluate the three questions posed in Section 1, we use a c implementation of Random, LRU and Clock replacement. This c file, named memsim.c evaluates the page fault rate, among other things, for a given trace file and a given frame amount. We then created a python file to run the memsim.c file, and report the page fault rate for every combination of file (gcc.trace, swim.trace, sixtrack.trace and bzip.trace), frame amount(all even integers from 2 to 100), and replacement algorithm (Random, LRU and Clock). The python file outputted the results of each experiment into a csv file. This csv file was then analysed with a jupyter notebook and matplotlib to create Figure 1.0 in Section 3. This figure depicts the page fault rate vs frame amount for each replacement algorithm, for each aforementioned file.

To understand how much memory each traced file needs, the frame amount which was the end of significant decline was identified. Significant decline denotes a change from one frame amount's respective page fault rate to the next frame amount's page fault rate, which was greater than the mean + one standard deviation. The last point in this significant decline set is the point in which any further frame amount increase results in minimal further page fault rate returns. Hence, this final point is the optimal frame amount for the given program. The results of this experiment are recorded in Table 1.0.

3. Experimental Results

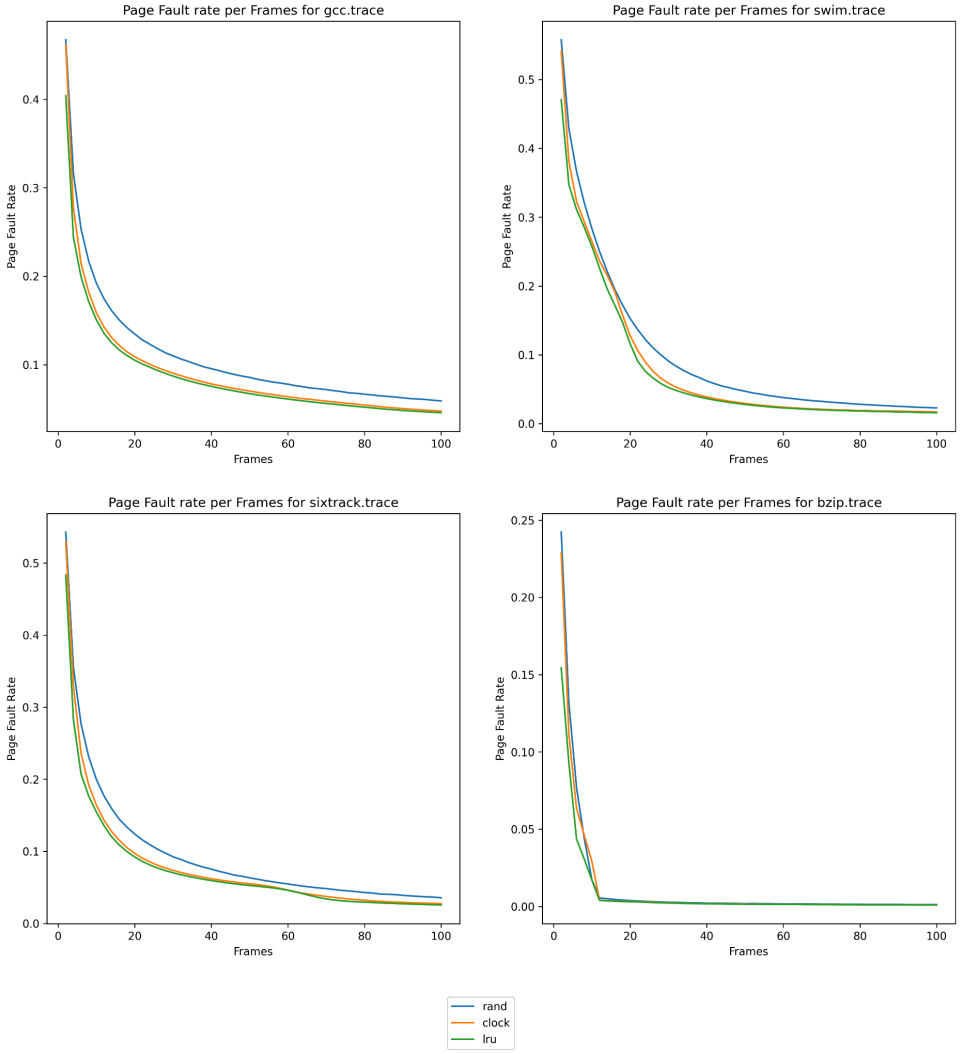

Figure 1.0, depicting the diminishing page fault rate as the frame amount increases for each application.

File					
		gcc.trace	swim.trace	sixtrack.trace	bzip.trace
Replacement Algorithm	LRU	22	40	26	14
	Clock	22	42	26	16
	Random	24	38	28	20

Table 1.0, showcasing the optimal frame amount for each application.

4. Results Analysis

gcc.trace

Visible in Figure 1.0, is the exponentially decreasing relationship between page fault rate and the amount of frames. We observe that LRU performs marginally better than Clock, before converging with a frame amount of 100; whereas Random replacement performs the worst with the highest page fault rate. When regarding Table 1.0, it is clear that LRU and Clock have the same optimal frame size of 22 frames. Comparably, Random replacement's optimal frame size only occurs one frame size larger at 24 frames. This indicates that GCC only needs around 24 frames at most for all tested replacement algorithms to perform well. Furthermore, when observing the page fault rate of each replacement algorithm for all frame sizes up until 10 we find that the replacement algorithms initially converge then diverge into the order specified above, highlighting that at frame sizes 2 and 4, there is little difference between each replacement algorithm's performance.

swim.trace

For swim.trace we find, through Figure 1.0, that the best performing replacement algorithm is less clear than the gcc.trace tests, as LRU and Clock converge with 36 frames of memory. Yet, Random replacement still performs worse. Notwithstanding, Clock and Random replacement converge at a memory size of 14 frames. In Table 1.0, Random replacement is shown to carry the lowest optimal frame size of 38 frames, compared to LRU and Clock with 40 and 42 respectively. This indicates that Random replacement was the fastest to reach a state of diminishing page fault rate returns. However, the high number of frames required to reach an optimal memory size indicates that swim.trace is a computationally expensive application to run in general. With a frame number less than 10, LRU provides the lowest page fault rate, followed by Clock and Random replacement.

sixtrack.trace

The performance of the three replacement algorithms in sixtrack.trace is similar to gcc.trace, however, LRU and Clock converge at 34 frames of memory, with random replacement still bearing the highest page fault rate. Interestingly, both Clock and LRU replacement have the same optimal frame amount of 26 frames, suggesting that Clock and LRU are equally effective in managing the memory of the sixtrack application. With less than 10 frames, the order determined in the last two files was

conserved. Yet, the three replacement algorithms remain converged from frame's amount of 2 to 6, highlighting that at a low memory size, the replacement algorithm is insignificant for Sixtrack.

bzip.trace

Finally, bzip has the most interesting results for the effect of frames in memory on page fault rate. As is visible in Figure 1.0, the page fault rate of each replacement algorithm reaches zero by a memory size of 20 frames. LRU is the first replacement algorithm to reach a zero page fault rate, with 14 frames, whereas Clock replacement converges to zero with 16 frames. Consistent with the trend of previous applications, Random replacement is the last to converge and hence performs the poorest of all three replacement algorithms. On frame sizes less than 10, LRU performs the best, with the lowest page fault rate, while Random and Clock replacement perform similarly. As Random replacement converges to zero last, bzip as an application only needs a maximum of 20 frames in memory.

5. Conclusion

Throughout this experiment we find that as the amount of frames available increases, the page fault rate is decreased. In Figure 1.0, we observe that LRU replacement consistently performs the best, however, at times converging with Clock replacement. For a low number of frames, the best page replacement algorithm is less clear, with most replacement algorithms bearing similar page fault rates. Nonetheless, LRU tends to perform better with a lower number of frames. Moreover, LRU is the first replacement algorithm to reach an optimal frame amount in three out of the four applications tested, further demonstrating its effectiveness as a page replacement algorithm. Finally, through Table 1.0 we find that gcc requires at least 24 frames of memory to perform optimally, whilst swim requires at least 42 frames of memory for optimal page fault rates. Moreover, sixtrack was found to require at least 28 frames of memory, whilst bzip was the least computationally expensive with at least 20 frames of memory to work optimally.

Bibliography

Aleks (2020) *Programs, Processes, and Threads - What Are They, and How Do Computers Run Them?*, *Medium*. Medium. Available at: https://medium.com/@al.eks/programs-processes-and-threads-what-are-they-and-how-do-computers-run-them-68f56ce023aa (Accessed: 14 September 2024).

Tyson, J. (2000) *How Virtual Memory Works*, *HowStuffWorks*. HowStuffWorks. Available at: https://computer.howstuffworks.com/virtual-memory.htm (Accessed: 14 September 2024).

What is Thrashing? Why Does it Occur? (no date) *Why Does it Occur?* | *Lenovo US*. Available at: https://www.lenovo.com/us/en/glossary/thrashing/?srsltid=AfmBOoqhP6icMjWtDq_CCJ4iVx5qJbYW2_I_OkWh7J9GjPtyp3JFYyMF (Accessed: 14 September 2024).

Memory Paging and Page Tables (no date) *CS 240: Introduction to Computer Systems (Spring 2021)*. Available at: https://courses.grainger.illinois.edu/cs240/sp2021/notes/paging/pageTable.html (Accessed: 14 September 2024).

Raza, S.. (2023) *Understanding Virtual Memory and Paging*, *DEV Community*. DEV Community. Available at: https://dev.to/syedmuhammadaliraza/understanding-virtual-memory-and-paging-3ope (Accessed: 14 September 2024).

GeeksforGeeks (2024) *Page Fault Handling in Operating System*, *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/page-fault-handling-in-operating-system/ (Accessed: 14 September 2024).

OS Project (no date) *Page Replacement Algorithm*. Available at: https://os-project-page-replacement.vercel.app/ (Accessed: 14 September 2024).

GeeksforGeeks (2023) *Second Chance (or Clock) Page Replacement Policy*, *GeeksforGeeks*. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/second-chance-or-clock-page-replacement-policy/ (Accessed: 14 September 2024).

Computer Science | *University of Illinois Chicago* (no date) *Operating Systems: Virtual Memory*. Available at: https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/9_VirtualMemory.html (Accessed: 16 September 2024).