Teaching and Learning Page Replacement Algorithms using a Java-Based Visualization Tool

SUKANYA SURANAUWARAT
School of Applied Statistics
National Institute of Development Administration
118 Seri Thai Rd., Bangkapi, Bangkok 10240
THAILAND
sukanya@as.nida.ac.th  http://as.nida.ac.th/~sukanya

Abstract: - This paper presents a Java-based visualization tool that uses graphical animation to convey the concepts of various page replacement algorithms. This tool is unique in a number of respects. First, it differentiates the read-access pages from the write-access ones, since the cost of replacing a page that has been modified is greater than for one that has not. Second, it allows the user to practice and test his understanding of the concepts he has learnt through a very easy-to-use graphical user interface. Third, it allows the user to compare the performance of two different algorithms or that of the same algorithm with different conditions in an easy manner. This tool can be used by teachers and students in Operating Systems courses as a teaching and learning aid or by anyone who is interested in page replacement algorithms.

Key-Words: - Algorithm Animation, Visualization Tool, Computer Science Education, Page Replacement Algorithms, and Operating System

1 Introduction
Most modern operating systems use paging for virtual memory memory management, where the process address space is divided into pages and can be assigned to the corresponding units in physical memory called frames or page frames. When a process references a page that is mapped into its address space, but not loaded into physical memory, a page fault is said to occur. When all of the frames in physical memory are occupied and it is necessary to bring in a new page to satisfy a page fault, a page replacement algorithm is used to determine which page currently in memory is to be replaced. Many page replacement algorithms exist and most of them try to guess which page to replace, based on the past behavior, to minimize the total number of page faults, while balancing this with costs of the algorithms themselves.

For many types of computer science algorithms, studies have shown that animation can do much to enhance learning and understanding [1][2]. Therefore, the author has developed a visualization tool called Page Replacement Algorithm Simulator that uses graphical animation to convey the concepts of various page replacement algorithms. The simulator is unique in a number of respects. First, in addition to the number of page faults, the simulator also evaluates the performance of a page replacement algorithm in terms of the cost of replacing pages, since the cost of replacing a page that has been modified is greater than for one that has not. This is because the former must be written back out to disk before the new page can be read in. The second unique feature of the simulator is that it allows the user to practice and test his understanding of the concepts he has learnt through a very easy-to-use graphical user interface. The author believes that this feature is important, since it has been reported that learners who are actively engaged with visualization technology have consistently outperformed learners who passively view graphics [3][4]. For example, Byrne et al. [5] conducted an experiment in which viewers were forced to make predictions about what they would see. These viewers scored significantly better on a post-test than others who merely watched identical animation without predicting what they would watch. The third unique feature of the simulator is that it allows the user to compare the performance of two different algorithms or that of the same algorithm with different conditions. By using this feature of the simulator, the user can explore under what conditions a page replacement algorithm performs well and under what conditions it is better than another algorithm.

The simulator is designed as a teaching and learning aid to be used by teachers and students in Operating System courses or by anyone who is interested in learning page replacement algorithms in an easier and a more effective way.
The remainder of this paper is organized as follows: section 2 is a brief overview of the page replacement algorithms used in the simulator, section 3 gives a description of the simulator, section 4 discusses related work, and section 5 draws some conclusions.

2 Overview

There are various page replacement algorithms. The simulator uses the algorithms listed below (which are discussed in [6][7]).

- **Optimal**: replaces the page that will either never be needed again, or will not be used for the longest period of time. This “unrealizable” algorithm is usually used only as a benchmark by which other algorithms can be judged.
- **First-In-First-Out (FIFO)**: replaces the page that has been in memory the longest. FIFO incurs low overhead but generally does not predict future page usage accurately.
- **Least Recently Used (LRU)**: replaces the page in memory that has not been referenced for the longest time. LRU generally predicts future page usage well but incurs significant overhead.
- **Most Recently Used (MRU)**: replaces the page in memory that has most recently been used.
- **Clock**: is a modified form of the FIFO algorithm. It treats the page table as a circular list of pages, and uses the reference bit associated with each entry in the page table and a pointer (the “clock hand”) into the page table. The reference bit is set whenever a page is referenced. When a page must be replaced, the algorithm checks the page table entry pointed to by the clock hand. If the referenced bit for that page is set, it is cleared and the clock hand is advanced. It continues in this manner until it finds an entry whose reference bit is off, and in that case it selects that page for replacement.
- **Enhanced Clock**: makes the clock algorithm more powerful by increasing the number of bits that it employs. That is, it makes page replacement decisions using two bits: the reference and modify bits. Whenever a page is referenced, the reference bit is set; whenever modified, the modify bit is set. When a page must be replaced, the algorithm begins with the page table entry pointed to by the clock hand. If the reference and modify bits for that page are set, the reference bit is cleared and the clock hand is advanced; if the reference bit is set but the modify bit is not, the reference bit is cleared and the clock hand is advanced; if the reference bit is clear but the modify bit is set, the modify bit is cleared (and the algorithm notes that the page must be copied out before being replaced) and the clock hand is advanced; if both the reference and modify bits are clear, the page in that frame is replaced.
- **Least Frequently Used (LFU)**: selects a page for replacement if the page has not been used often in the past. This algorithm keeps a counter of the number of references that have been made to each page. Pages with the lowest counts are replaced while pages with higher counts remain in primary memory.
- **Most Frequently Used (MFU)**: replaces the page with the largest count. This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

All of the policies have as their objective that the page that is removed should be the page least likely to be referenced in the near future. Therefore, an important measure of goodness for a page replacement algorithm is the number of page faults generated for a given reference string (i.e., a sequence of page numbers referenced by a process during its execution) and number of frames.

3 Design and Features

The simulator is written using Java 6. It has three operating modes: simulation, practice, and comparison modes. In simulation mode, the user can watch virtually step-by-step how an algorithm works or watch it run straight through from the beginning until the end. In practice mode, the user can decide which page replacement algorithm he wants to test his understanding of, and then he can try to select the pages that will be replaced by himself through a very easy-to-use graphical user interface after that he can check whether his answers are correct or not with the simulator. In comparison mode, the user can compare the performance of two different algorithms or that of the same algorithm with different number of frames in an easy manner. Each mode is described below.

3.1 Simulation Mode

Figure 1 shows a snapshot of the simulator during a simulation in simulation mode. The top panel of the screen allows the user to control the simulator as he desires; specifically, the user can select which
algorithm to animate through a drop-down list box located at the top of the panel. When the user selects the algorithm, the user can also specify his own reference string and the number of page frames, or he can use the default values.

In Figure 1, the state of the simulator at virtual time 3 is shown during a simulation using the optimal algorithm with the reference string of: ‘7* 0 1* 2 0 3 0 4* 2* 3 0 3 2 1 2 0 1 7 0 1’ and the number of frames set to 3. The virtual time of a process represents the progress the process makes as it executes. In other words, the virtual time is advanced each time the process makes a memory reference. The simulator uses the blue highlight to indicate which page is currently referenced. As shown in the middle panel of the screen, page 2 is referenced at virtual time 3. Since page 2 is referenced for the first time and all of the frames – three frames, in this example – are occupied, it is necessary to replace one of the current three frame pages, the victim page, with page 2. When this happens, the message “A page fault has occurred” followed by a second message “Selecting a victim page” are displayed. Then the animation shows how a victim page is selected to be replaced under the optimal page replacement algorithm. That is, it scans the rest of the reference string starting from the page referenced during the next virtual time (i.e., virtual time 4) until it determines which page is referenced latest or never; that one is selected as the victim page. The scanning process is shown by a moving arrow above the reference string. During the scan, it found that pages 0, 1, and 7 are referenced again at virtual time 4, 13, and 17, respectively, as shown by the green highlights. After finding the next reference for each page, it ends the scanning process at virtual time 17 concluding that page 7 is the victim page. Once the victim page is determined, the frame in the middle panel that has that page will be highlighted in red and will blink a couple times to draw the user’s attention. After that, the simulator will replace the victim page with the new page and then display the message “The page fault cost is 2”. At this point, the table in the bottom panel will be updated with the data from virtual time 3. Then virtual time will increment to virtual time 4. Note that the page fault cost is calculated based on whether the victim page had been modified or not. If the victim page is modified, the total cost of a page fault is 2 I/O transfers, while the cost of an unmodified victim page is 1 I/O transfer. The cost of replacing a page that has been modified is greater than for one that has not, because the former must be written back out to disk before the new page can be read in.

The main feature of the bottom panel is a table. The table has a row for the request and a row for each frame, as well as, a column for each virtual time t. A table entry at Frame i and column j shows the page loaded at time t in Frame i after r_j (which is the

![Figure 1: A snapshot of the simulator during a simulation using the optimal algorithm.](image)
the number of the page referenced by the process at its virtual time \( j \) has been referenced. If the entry is highlighted with a different color, the page shown in the entry was loaded as a result of the missing page fault. Each column heading is a virtual time; under the headings, there are pages from the reference string. Below the table, the total number of page faults and the total cost of page faults that have occurred so far are displayed. Note that, only page faults occurring after the frame allocation is initially filled are counted.

The next algorithm to be discussed is the LRU algorithm. When the LRU algorithm gets to virtual time 3 with the same reference string that was used with the optimal algorithm shown in Figure 1, instead of looking ahead the algorithm looks back to determine which page will be the victim page. This is animated by the simulator in the same way as described for the optimal algorithm, except it highlights the prior occurrence of each page before the current virtual time instead of the future occurrence of the page. Once the least recently used page is determined, it becomes the victim page which is replaced by the current reference string page. After that, the table in the bottom panel will be updated with the data from virtual time 3. Then virtual time will increment to virtual time 4.

The MRU algorithm is also animated by the simulator in a similar way to the LRU algorithm except that it highlights only the most recently occurrence of the page prior to the current virtual time.

The rest of the algorithms, which are the enhanced clock, the clock, the FIFO, the LFU, and the MFU algorithms, are very similar in their simulation. For the enhanced clock algorithm, the page in each frame is associated with a reference bit (R bit) and a modify bit (M bit); and the pointer (arrow) is set to the oldest page, as shown Figure 2. When the enhanced clock algorithm gets to virtual time 3 with the same reference string that was used with the optimal algorithm shown in Figure 1, a page fault occurs. It then scans R and M bits of the page in each frame to find a page that has both R and M bits set to zero (i.e., R and M bits are off); this is simulated with a moving highlighted region as in Figure 3. During the scan, each time it encounters a page that does not have both R and M bits set to zero, it turns off one of these two bits according to the rule described in section 2 and continues on. Once it encounters the oldest page with both R and M bits set to zero – page 0 in this example – the frame in the middle panel that has that page will be highlighted in red and will blink a couple times to draw the user’s attention. After that, the simulator will replace the victim page with the new page following the same process described early for the optimal algorithm.

The clock algorithm is also animated by the simulator in a similar way to the enhanced clock algorithm except that it uses single-bit reference information to make page replacement decisions. Hence, the second column of the table in the middle panel displays only the R bit of each page.

The rest of the algorithms, the FIFO, the LFU and the MFU algorithm, run in a similar way to the enhanced clock algorithm except that there is no pointer set to the oldest page and the information in the second column of the table is different. In the
case of the FIFO algorithm, the second column is the time each page is loaded and in the case of the other two algorithms, LFU and MFU, the second column of the table is a counter (i.e., a counter of the number of references that have been made to each page). When a page fault occurs, the FIFO algorithm will scan for the page with the oldest loaded time; whereas the LFU and the MFU algorithm will scan for the page with the smallest and the largest counts, respectively. Note that the numbers displayed after “/” in the table at the bottom panel of Figures 2 and 3 represent the reference or modify bit value of each page and the symbol “<” in the same table represents the current position of the pointer.

3.2 Practice Mode
In Figures 4 through 7, different snapshots of the simulator in practice mode are shown. As in the simulation mode, the user needs to specify which page replacement algorithm will be used, as well as a reference string, and the number of page frames; otherwise, the default values will be used. In practice mode, there are two ways of practicing: One-step-at-a-time and All-at-once. By clicking on the corresponding radio button located under the “Random” button, the user can choose which way to practice. Both ways of practicing are described in detail below.

![Figure 4: A snapshot of the simulator at virtual time 3 in One-step-at-a-time practice mode.](image)

Figure 4 shows a snapshot at virtual time 3; at this point, the simulator is waiting for the user input about virtual time 3. The graphic in the middle indicates the user input for virtual time 2 was correct; if the user input was incorrect then the graphic would say it was wrong as well as highlighting the incorrect answers in the right table. Note that a correct answer for the clock algorithm includes the reference bit value, and for the enhanced clock algorithm includes both reference and modify bit values. Also, for the sake of convenience, the user does not need to type the “*” that is used to indicate that the pages are write accesses. After indicating if the answer is correct the data in the left table is updated to reflect the correct data. In the table at the bottom of the panel the current virtual time is highlighted in green and the page faults are highlighted in red. In Figure 5, the user has inputted data for virtual time 3 in the left table and clicked the “Submit” button; the simulator has acknowledged the answers are correct and the data in the tables is updated.

![Figure 5: A snapshot of the simulator at virtual time 4 in One-step-at-a-time practice mode.](image)

![Figure 6: A snapshot of the simulator in All-at-once practice mode.](image)
Figure 6 is a screen shot of the simulator in All-at-once practice mode. In All-at-once practice, the user is expected to enter all the data into the practice table at the bottom for all virtual time. Once the user has input as much data as desired, clicking the “Submit” button will cause the simulator to highlight any incorrect answers and enable the “Display Answer” button. When the user clicks the “Display Answer” button, the correct answers will be displayed in the practice table as shown in Figure 7.

![Figure 7: A snapshot of the simulator in All-at-once practice mode after the “Display Answer” button is clicked.](image)

3.3 Comparison Mode
A page replacement algorithm can be evaluated by running it on a particular reference string, and computing the number of page faults and the page fault cost. The fewer the number of page faults and the lower the page fault cost, the better the performance is, and the better the replacement algorithm is. In comparison mode, the user can compare the performance of two different algorithms. The user can also compare the performance of the same algorithm with different number of page frames as shown in Figure 8, which is an example of Belady’s anomaly, the phenomenon where adding more page frames does not reduce the number of page faults. Moreover, the user can use this mode to explore the effect of a pattern of page accesses on different algorithms. For example, the user can explore which pattern of page accesses for which MRU performs better than LRU, or vice versa.

![Figure 8: A snapshot of the simulator in comparison mode.](image)

4 Related Work
Many tools have been devised to aid the study of page replacement algorithms that are used in the context of memory management. Most of the tools focus on demonstrating page replacement algorithms by running the algorithms with various reference strings and a varying number of page frames, and then computing the number of page faults. These tools usually produce results similar to those shown in the table in the bottom panel. However, these tools usually differ in the look-and-feel and the number of the algorithms they support which usually range from 2 to 5. Also, most of them do not distinguish whether the pages in the reference string are read or write accesses, so they do not calculate the page fault cost and do not support the enhanced clock algorithm. Some examples are those tools implemented by KIrsa1 Koren [8], Samir Solanki [9], Qi Song [10] and Du Zhang [11]. Since the main goal of the author’s tool is to use graphical animation to convey the concepts of various page replacement algorithms, the length of the reference string and the number of page frames the user can specify are more limited.

Although, PAGE [12] has a similar goal to the author’s, it shows how each algorithm works based on how the operating system implements the algorithm. For example, with FIFO algorithm, the operating system maintains a queue, with the oldest page at the head of the queue and places the most recent arrival at the tail of the queue. When a page must be replaced, PAGE removes the page at the head of the queue, and adds the new page at the tail of the queue. Another example is the clock algorithm which can be implemented by using a
circular list of pages in memory, with the clock hand pointing to the oldest page in the list. When a page fault occurs and no empty frames exist, then the R bit is inspected at the hand’s location. If R is 0, the new page is put in place of the page the clock hand points to; otherwise, the R bit is cleared. Then, the clock hand is incremented and the process is repeated until a page is replaced.

Finally, none of the existing tools provide any function that is similar to the practice and the comparison modes of the author’s simulator.

5 Conclusion
This paper presents a simulator that uses graphical animation to convey the concepts of various page replacement algorithms that are used in the context of memory management. The simulator supports various types of page replacement algorithms. There are three operating modes for the simulator; the first is simulation mode, the second is practice mode, and the third is comparison mode. In simulation mode, the user can start and stop the simulation whenever he wishes, and watch the simulation straight through from the beginning until the end, or watch it step-by-step. By using the simulator in simulation mode, the user could achieve a better conceptual understanding of the page replacement algorithms. In practice mode, the user can test his understanding in two ways: One-step-at-a-time and All-at-once. In comparison mode, the user can experience Belady’s anomaly and can explore under what conditions a page replacement algorithm is better than another algorithm.

Future work will include an extensive experiment with the simulator in a computer laboratory to determine its effect on student learning.

References: