

CSE 312

OPERATING SYSTEMS

Final Project Report

Lecturer : Prof. Dr. Yusuf Sinan Akgül

Gökhan Has - 161044067

July 10, 2020

CSE 312 OPERATING SYSTEMS FINAL REPORT

Page Entry Structure in Page Table

In virtual memory, pages are placed using a table that indexes the memory. This structure is called "page table". The page table held in memory is indexed by the number of the virtual memory address and contains the corresponding actual page number. Each program has its own sheet table that converts the virtual address space into the memory space in the main memory. The page table can also keep records of pages that are not available in the main memory. The current bit (1 or 0) is kept in each page table. If this bit equals logical zero, it means that the page is not available in the main memory and a "page fault" occurs. If the bit points to a logical one, the page has a valid physical address available in the main memory.

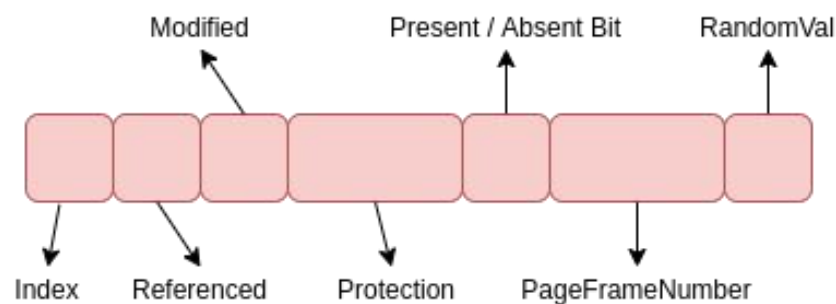


Figure 1 : Page Entry

Index : It holds the index in the page table. The reason for this is to save time by setting references in ranking algorithms.

Referenced : The reference bit is to assist some of the page replacement algorithms. It needs to be evacuated without memory error. Some algorithms use this bit.

Modified : The modified bit is actually set on the hardware side. However, I had to adjust it myself as I could not adjust the hardware ourselves. If the page is changed, it is written back to disk. This bit is used to see if it has changed. Some page replacement algorithms also use this bit. Care was taken when implementing algorithms.

Protection : Protection is kept as a string. Although it is not used much in the program, it was kept because it is normally necessary. Used in normal systems for page protection.

Present / Absent Bit : Present / Absent bit. If this bit is 1, the input is valid and can be used. If it is 0, the virtual page to which the entry belongs is not currently in memory. Accessing a page table entry with this bit set to 0 causes a page fault.

Page Frame Number : The most important field is the Page frame number. After all, the goal of the page mapping is to output this value. It is used to calculate where the data to be sorted is located on the disk. It is important. If there is a wrong value, the order is not correct.

Random Val : The random value is assigned to -1 in the fill function. As the page replacement is done, it is taken with the get function and assigned to the entry. (The -1 assignment is made on part2.cpp lines 870 and 899.)

Page Replacement Algorithms

1- Not-Recently-Used (NRU) :

R and M bits are used in this algorithm. When there is a page fault, that is, if the values of that page are not found in physical memory, the operating system would normally catch it and change the pages according to the four zeros. But since we don't have the chance to do this, I solved it with the help of a function. (line 1044 at part2.cpp and getNRUClassNumber () function.) From there you can return 4 classes. Their number is 0,1,2 and 3, respectively. 0-> not referenced, not modified, 1-> not referenced, modified, 2-> referenced, not modified, 3-> referenced, modified. If class 1 and class 0 have it, that page is changed. Normally, R bits are reset every time an interrupt occurs. However, since there is no situation like interrupt capture, I applied a solution such as resetting the R bits when every sort algorithm changes. Class 0 means 1 less used here. This algorithm is easy to understand, efficient and has sufficient performance.

Page fault operation in the NRU is done within the get method. part2.cpp starts at line 454. A list is used for this. This list only holds the indexes on the page table. In the 462th line, the class number is obtained with the help of the getNRUClassNumber () function. Then the check is made, if it is not 0 or 1, the list continues. If no suitable value is found, it is preferred to take the first value and continue it. Otherwise, a situation that takes a lot of time occurs. Then the value is taken from the disk, the page table is changed with the help of the set function. Of course, the value of physical memory is brought before this. Then the index is added to the list again and the return method ends by returning the value.

2- First In First Out (FIFO) :

According to this algorithm, when there is a page fault, that is, when a page that is not in the physical memory is wanted to be accessed, that is, when a page on the disk is wanted to be accessed, while the related page is loaded into the physical memory, the oldest page in the memory is loaded back to the disk and this oldest page is also written back to the disk.

For the FIFO algorithm, the queue data structure was used in accordance with the FIFO principle. It starts from 332th line in part2.cpp. The oldest page index is popped, then removed from the disk. With the set function, memory is accessed and physical memory is also thrown. Then the new page index is pushed to queue again.

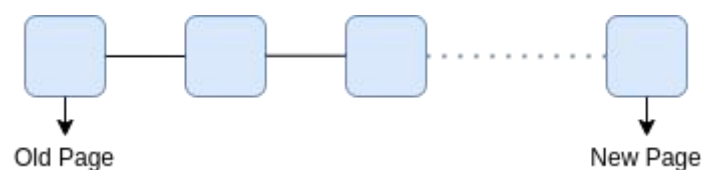


Figure 2 : FIFO Algorithm

3- Second Chance (SC) :

FIFO page replacement algorithm modified form, known as the second chance page replacement algorithm, is relatively better than FIFO fees for a small fee for improvement. It does FIFO, but instead it works right through paging that page, looking in front of the queue as if it checks whether its referenced bit is set. If it is not set, the page is swapped. Otherwise, the referenced bit is cleared (as if a new page) is placed behind the page queue and this process is repeated. This can also be considered as a circular row. If all the pages have been set up a bit of their referenced, now on the second encounter of the first page in the list, that page will be swapped, as their referenced has been cleaned up a bit. If all pages have received their reference bits, then the second chance algorithm turns into pure FIFO.

In this algorithm, the R bit of the oldest page is examined. 0 means that the page is old and unused. So it can be changed. If R bit is 1, the bit is cleared and put at the end of the list. The loading time is updated as if it is new to memory. Then the call continues. Thus, the second chance is given. But this algorithm may have problems in the worst case. Because if all R bits are 1, all pages will be given a second chance and the algorithm will turn into FIFO. SC operations begin on line 597 in part2.cpp. I used the queue data structure I wrote here. Because holding the R bits twice also speeded up my code and program. It was more effective. My aim was to ensure the correctness of the code by doing parallel control.

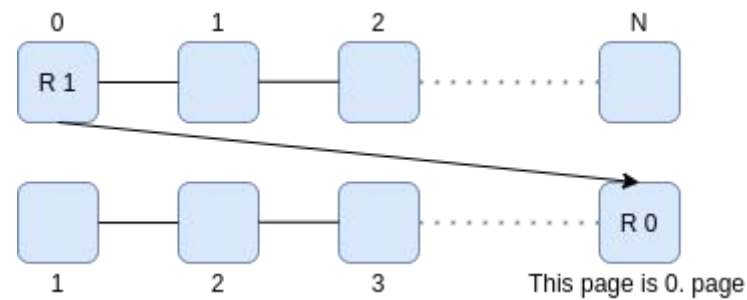


Figure 3 : Second Change Algorithm

4- Least-Recently-Used (LRU) :

While the most used (LRU) page replacement algorithm is similar to its name for NRU, the fact that LRU tracks page usage over a short period of time is different, while NRU only looks at usage over the past hour range. LRU is working on the idea that the pages that have been used in the last few instructions are very likely to be used heavily in the next few instructions. LRU (almost like it can provide near-best performance in theory) is very expensive to implement in practice. Try to keep the cost as much performance as possible yet. There are several implementation methods for this algorithm. The most expensive method memory is a linked list method that uses a linked list containing all pages. The most recently used page behind this list and the most recently used page at the front. It lies in the fact that each memory reference, which is a very time consuming process on the application cost list, will have to be moved around.

The hardware is increased with each instruction There is a 64-bit counter, I suppose: Another method that requires hardware support is as follows. When a page is accessed, the counter gains equal value while accessing this page. When a page needs to be replaced, the operating system selects the page with a low counter and replaces it parts. With the current hardware, this is not possible because it is necessary to examine the counter for each page in the OS cache.

Because implementation costs are similar in a single LRU, but algorithms which offer cheaper applications (such as followers) may consider.

We stated that we did not use hardware in this project. That's why I found it appropriate to use timestamp. I'm looking at the times with the timestamp when the page is created or when the new page arrives. Thus, I have simulated the aging phenomenon. I'm using the `getLRUClockTime()` function to return the oldest page index. Normally 1 bit right in counter shifting and R bit was added. Then the R bit was added to the leftmost bit. Of course, this process was received on the hardware side and was called aging. `part2.cpp` 731th line these operations are carried out.

5- Working Set Clock (WSClock) :

The WSClock algorithm is used because the WS algorithm is bulky. Because in WS algorithm, the entire page table is scanned. This is not the case with this algorithm. That's why it's fast. It is a combination of Clock + WS algorithms. In a list, indexes are kept. I created the clockQueue structure by changing the queue production I wrote. I continued my transactions through this structure. Here I created nodes to hold both the last element, the first element, and the current element. So I can do the roaming. I kept the current virtual clock and R bit in the list, that is, clockQueue. These were held to ensure security and to be fast. Otherwise there was no need to be kept. If the value of the R bit is equal to 0 or the age is larger, a new page should be brought. If the page has a modified bit of 1, it must be written back to disk. For this, it is requested to write back to disk. And the current node is set to show the next one.

It can write all of them to disk. This is the bad feature of this algorithm. So we talked in the lesson that it was necessary to set a limit value. But I did not set a limit value. If it returns to the beginning, the original page will be returned. If a full tour is taken and comes back to the beginning, I came across the following scenarios. First, the current node continues to move. One of the writings will of course end and that page can be removed. Secondly, if all the passwords are not used, the value that the current head holds is selected as the victim and written back to the disk. There are codes from line 801 on part2.cpp. Likewise, the clockQueue.cpp file can be checked.

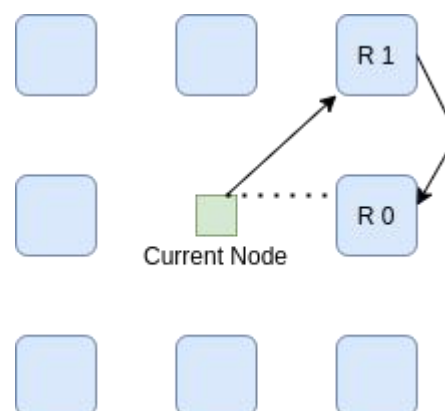


Figure 4 : Working Set Clock Algorithm

Local vs Global Allocation

I searched for an answer to the question of whether the current processes' page will change or one of the global processes' page will change. We argued that it is better to delete globally in general. However, in one approach, giving an equal number of pages to all processes was advocated. But this approach was unfair. Therefore, it is more fair to give each transaction proportional to its total size. However, it is not possible to know the total size of the transactions. So I had to give the same number of pages to all processes. If it is local, I just changed the password of my own process. But there were problems here. Destruction problem, trashing problem and waste memory problem.

Similarly, PFF should be considered. Page Fault Frequency tells you when to increase or decrease a process's page allocation. However, if the page fails, it does not say anything about which page to replace. Therefore, it was not preferred in the codes I wrote. Also, the WSClock algorithm works only locally.

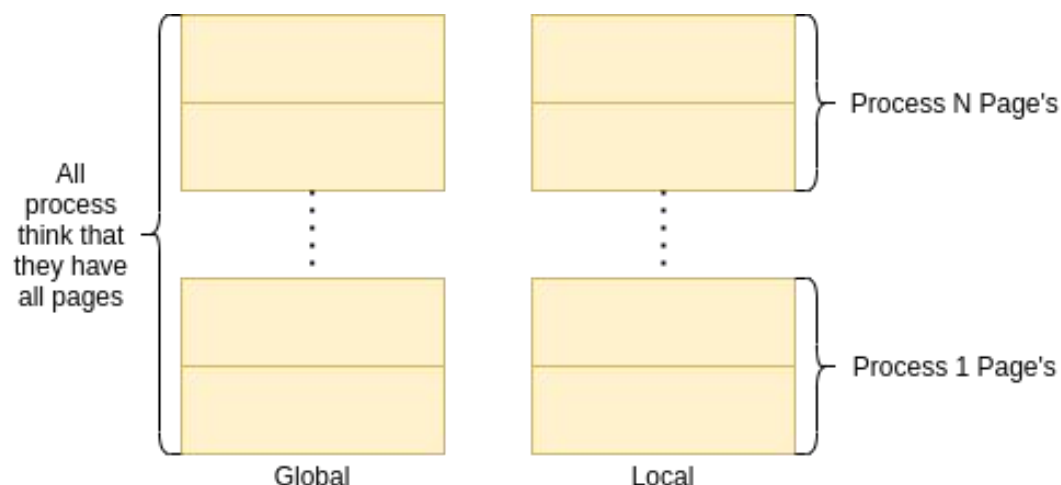


Figure 5 : Global vs Local Allocation

Set Function

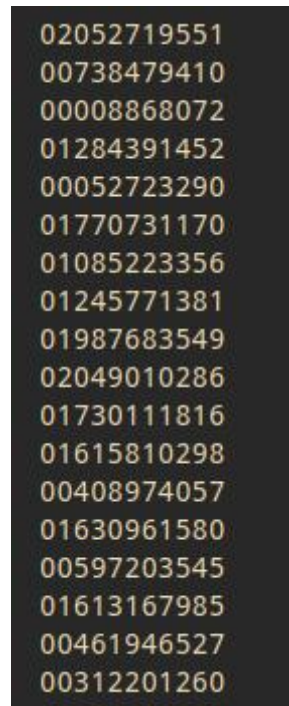
The set function is used to access memory. In page replacement algorithms, in the case of page fault, get is called inside the function. After the value in the memorizer is taken from the disk, a new value is set with the help of the set function. Likewise, writing a value to the disk file is written with the set function. The set function calls the helperSet function according to the tName parameter. The helperSet function does the above in the set function. Separate functions are called for fill and check operations.

Get Function

The Get function is one of the most important functions. It is the longest function implemented. It starts at line 305 in part2.cpp. In this function, firstly the page in present / absent bit is checked. If this is 1, it already exists in physical memory. So it is returned immediately. But if there is not any else is passed. Here, page replacement algorithms are said to be applied. It enters the specified block according to which algorithm is applied or global local change. After that, it changes according to the type of page replacement algorithm. The algorithms are described on the above pages. In summary, the necessary information is taken from the disk. But before that, the page to be removed is determined and needs to be removed. Then it is written to physical memory and the last value is returned. The returned value is written to the page table with the help of the set function.

Disk File And Backing Store

Random value values are kept in the disk file. There are 12 characters in each line to get data from disk. Thus, the fseek method was read at file O (1) time. This process is important because speed is important in sequencing algorithms. Because when there is a page fault, disk is used. It is a part of the hard disk that is not available in physical memory at the moment and is used by paging to store data. Since the number of page faults increased as a result of my trials, an effective algorithm had to be designed.



```
02052719551
00738479410
00008868072
01284391452
00052723290
01770731170
01085223356
01245771381
01987683549
02049010286
01730111816
01615810298
00408974057
01630961580
00597203545
01613167985
00461946527
00312201260
00070650000
```

Figure 6 : Disk File

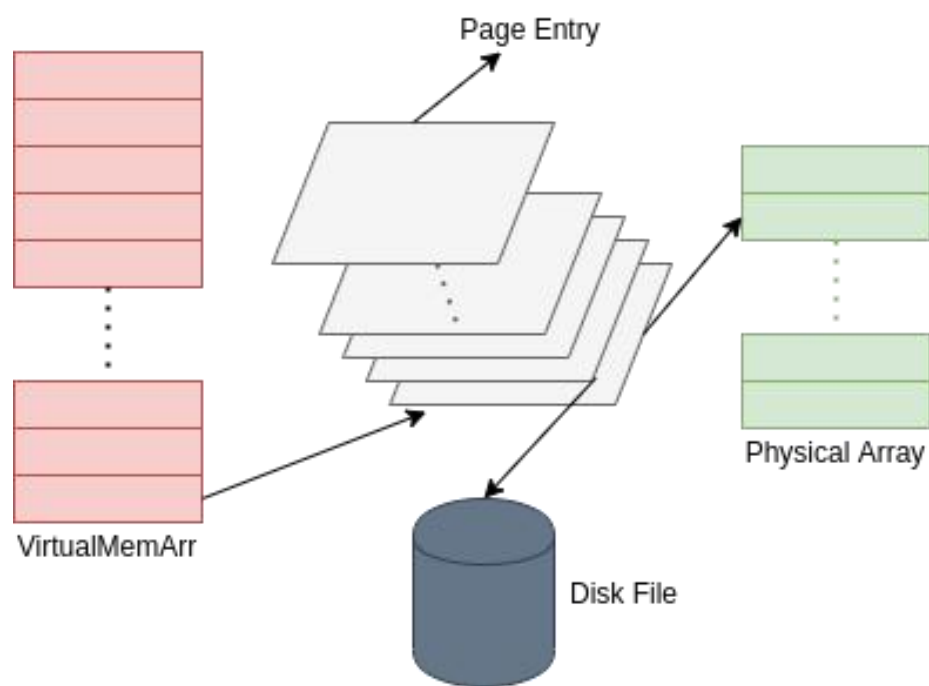


Figure 7 : All Program