Research on Replacement Algorithm of Dynamic Page Storage Management

Yongchao Zhao*, Ying Xie and Haifei Qin
School of Information Science & Technology, Chuxiong Normal University, China

*Corresponding author email: zhaoyongchao@cxtc.edu.cn

Abstract. This paper takes the dynamic page storage management in OS as the research aspect, and discusses the common three algorithms of FIFO, LRU and OPT. And put forward the idea of "vertically and horizontally", use pseudo code to implement these three algorithm. Although the algorithm can be improved, but our idea of unifying the three algorithms is of great significance.

Keywords: Replacement algorithm; Vertically; Horizontally; Page storage management.

1. Preface

Storage management in computer operating system is one of the important functions of OS. The page memory management in storage management is an important leap from the partition storage managed by continuous storage to the discontinuous storage. It not only realizes the discontinuous storage, but also solves the "fragmentation" problem of memory usage. For the storage management of page memory manager, the virtual memory(logical space) is divided into equal units, each unit is determined by the physical storage size. In general, its exponential power is 2, and it generally uses an integral multiple of 1K (e.g., 1K, 2K, 4K, etc.). Each of these cells is called a "page" and physical memory is divided into same size. In order to distinguish logical spaces, each unit of physical size is called a "block", and then logic and physics can be corresponded.

In virtual page storage management, the use of storage space is to load logical pages into some areas of memory, so as to achieve a multichannel environment. Because of the randomness of the page usage of the user's logical space, it is not transferred into the physical storage in order. In addition, according to the principle of locality, the logical pages divided by the user's logical space do not need to be fully loaded into memory to ensure the operation of the program. Therefore, it is not necessary to allocate the number of physical blocks corresponding to all the logical page numbers for all the pages needed by each logic program, but to load one or zero pages, and then load other pages dynamically according to the needs of the process operation; When the memory space is full, and new pages need to be filled in to enter the memory physical environment, a page is eliminated (or replaced) according to some algorithm, so as to load the new pages into the per divided physical blocks. This is the page replacement or elimination algorithm to be studied in this paper.

In the mapping process from logical address to physical address, it will implement according to a data structure called "page table". The relationship between logical page number and physical block number is the main record item of the whole page table’s records. During address mapping, if a page visited in the page table is not in memory, it generates "page missing" and an interruption. When page-interrupt, the operating system must select a page in physical memory to move it out of memory in order to make room for the page to be transferred into memory. However, in every page missing, a replacement strategy that seems to adopt the "lottery fair" method is to use "random" method to select
Choosing the page that is not often used to exchange out make the system better, which is also like the real-life purchase goods, the old (long-term) goods are popular. That is often not high. And the local principle of program running in the computer also tells, in most cases, after a frequently used page is accessed, then it is likely to be used again. If a page that is swapped out of memory, soon, it is swapped in. This kind of swapping in and out bring unnecessary system overhead, which is also called "shaking". Many kinds of algorithms for this kind of page replacement, and the typical are "First_in First_out (FIFO)", "Least Recently Used (LRU)", "optimal (OPT)". In common literature, gives the definitions of these 3 algorithms, and lists some examples to illustrate them. There is no unified understanding and grasp of them. In this paper, study a unified idea to look at these algorithms, and focus on the understanding of them from a unified point of view to realize the FIFO, FRU and OPT. The algorithm idea of "vertically and horizontally" is put forward to realize the three page replacement algorithms, which are described in the form of pseudo code and verified by examples.

2. FIFO Algorithms

In the real world, the idea of first in, first out can be seen everywhere. The check-out method of supermarket cash register customers and the on-site queuing and ordering method of fast food restaurants are all realistic descriptions of FIFO. That is, the customers who settle the account or order the meal first will leave the queue. This method adapted to in page storage management too. That is, in case of lack page in Memory, always select the first loaded one, or eliminate the longest time from the beginning to the obsolescence point, so as to replace the new in.

It is universal and easy to implement FIFO algorithm. The page numbers of the pages loaded into memory can be queued according to the order of entry, and the page numbers of the new pages can be queued at the end of the queue when a new page is loaded. When a page shortage occur, the first page of the team is always eliminated and the new page is added to the end.

We can think about this algorithm in this way. Because partial load memory is adopted, that is, the page number part required by the whole process is loaded into the physical block of memory, and the physical block allocated to each process is less than the total sequence number of the process. For example, a process needs pages 1-9 for different times, while only 3-4 physical blocks are allocated to the process to implement multichannel environment. In and out of the 3-4 that have been earned, one will be eliminated. Therefore, consider to measure these physical blocks at each elimination. Arrange the memory of each physical block in order, as shown in the following example.

For a page storage management, the page sequence needed is 3, 2, 4, 1, 4, 3, 5, 4, 3, 2, 1, 5, and the physical block obtained is 3. According to the FIFO idea, the replacement is shown in Table 1:

| No. | I | II | III | IV | V  | VI | VII | VIII | IX | X  | XI | XII |
|-----|---|----|-----|----|----|----|-----|------|----|----|----|-----|
| As  | 3 | 2  | 4   | 1  | 4  | 3  | 5   | 4    | 3  | 2  | 1  | 5   |
| Ma  | × | √  | √   | ×  | √  | ×  | √   | ×    | √  | ×  | √  | √   |
| Pr  | √ | √  | √   | √  | √  | √  | ×   | √    | ×  | √  | √  | √   |

As sequence.

Memory allocation.

Page fault. √ true, × false.

In Table 1, there are three situations: the first is that the memory at the beginning is empty, there is no page, and the page to be accessed has a storage location in memory. This situation can be put in order. In the second case, there is no page to visit in memory, and the memory block has been allocated, so one page needs to be eliminated according to the elimination algorithm. In this case, we need to eliminate the first memory according to FIFO algorithm. The third situation is that there are pages visited in the memory, and there is no need to exchange in or out. For these three cases, the first one is relatively simple. We mainly discuss how to apply the so-called "vertically" implementation idea we put forward in the others. For the second case, refer to table 1 for IV, VI, VII, VIII, X, XI, XII
invoking times. The third case is also special, refer to table 1 for V, IX. For the second, the consideration is that, according to the physical block obtained, it can use array or queue and other structural forms to store the visited page number, and the first page number stored will be eliminated in case of obsolescence. In Table 1, since fill in the latest page number from the bottom to the top each time it count in memory, it need to eliminate the top page number in the column where the table is located. This is the so-called "look up" idea. Of course, if using the situation shown in Table 2, the bottom page will be eliminated every time. This is the so-called "look down" idea. So it can be called "vertically" thinking.

Table 2. A FIFO Example of down mode.

| No. | I | II | III | IV | V | VI | VII | VIII | IX | X | XI | XII |
|-----|---|----|-----|----|---|----|-----|------|----|---|----|-----|
| As  | 3 | 2  | 4   | 1  | 4 | 3  | 5   | 4    | 2  | 1 | 5  |      |
| Ma  | 3 | 2  | 4   | 1  | 4 | 3  | 5   | 4    | 2  | 1 |    |      |
| Pr  | √ | √  |     | √  | × | √  |     |      | √  | × | √  | √   |

a Access sequence.
b Memory allocation.
c Page fault. √ true, × false.

From table 2, there are still three situations, which correspond to the column of Table 1 one by one. So it can extract the following core code for this Algorithm 1:

```java
int[] memList; // Represents a sequence of stored memory.
int[] arrayList; // Access sequence.
boolean lackPage; //Indicates no page to visit and space for new pages.
boolean replacmentPage; //Indicates that the page needs to be eliminated.
for(int i=0;i<arrayList.length;i++)
{
    if(lackPage)
    {
        add(newPage); //Add new pages accessed in turn.
    }
    if(replacementPage)
    {
        pop(memList[0]); //Eliminate the first page number.
        add(newPage); // New page add in memory.
    }
}
```

Algorithm 1. FIFO algorithm’s pseudo code and annotation.

Through verification, it gets that the number of page fault counts in this example is 10 times. According to the formula of page missing rate \( P = \left( \frac{count}{length} \right) \times 100\% \), the rate of missing pages is 83.3%. The result is consistent with the idea of FIFO definition.

3. LRU Algorithms

The idea of least recently used page replacement algorithm is: the pages that are used more frequently in the front are likely to be used more frequently in the back; conversely, the pages that have not been used for a long time are likely to not be used for a long time in the future. That is, in case of missing pages, eliminate the pages that have not been used for the longest time. LRU algorithm always selects pages that have not been visited in the longest time.

There are many algorithms about LRU algorithm. One is to add a "timing" flag to each page in the page table to record the time that the page has experienced since it was last visited, and the timing shall be restarted from "0" for each visit. When a new page needs to be loaded, check the timing flag of each page in the page table, select the page with the largest timing flag to call up (that is, the page that has not been used for the longest time in the latest period of time), and set all the timing flags of each page to "0", and then count again. When a page shortage occurs again, it can find the most recently used page to replace. This algorithm must record and update every page's visit situation all the
time, which is troublesome and expensive to implement.

Another algorithm is that there must be a value field enough for each instruction executed in each page table item. After each access to memory, the counter value is kept in the page table to be accessed. In case of page missing, check all the counter values in the page table, select the smallest page which is the most recent unused page, and exchange it out. This algorithm needs special hardware implementation.

Here the idea is that the LRU algorithm can take the current time as the evaluation time point and look to the left side of the access sequence. The distance to the left side of the current access sequence, which is the farthest from the current distance, is replaced when it is eliminated. Because every time think from the left side of the sequence point, it call "look left" in order to connect with the "look up and down" of FIFO. The following Table 3 is an example below of the above access sequence for instance.

Table 3. A LRU Example of left mode.

| No. | I  | II | III | IV | V  | VI | VII | VIII | IX | X  | XI | XII |
|-----|----|----|-----|----|----|----|-----|------|----|----|----|-----|
| As  | 3  | 2  | 4   | 1  | 4  | 3  | 5   | 4    | 3  | 2  | 1  | 5   |
| Ma  | 2  | 2  | 2   | 3  | 3  | 3  | 3   | 3    | 3  | 3  | 5  |     |
| Pf  | √  | √  | √   | ×  | √  | ×  | ×   | ×    | √  | √  | √  |     |

a Access sequence.
b Memory allocation.
c Page fault. √ true, × false.

From Table 3, it can be observed that there are still three situations in the access sequence: the first is that there is no page to be accessed in the memory, and there is enough space in the memory to accommodate the new page, at this time, the visited page is still directly added to the memory. The second situation is that there is no page memory to be accessed, and there is a page shortage. You need to swap out a page in memory to put a new page in. The third situation is that there are pages to be visited in memory that do not need to be replaced and do not need to move the memory to store the page location (of course, when using the stack to implement an algorithm, it is necessary to move the memory location to keep the top of the stack is the most recently visited page and the bottom of the stack is the most recently unused page). The focus, of course, is on the second. When the second situation occurs, it is considered according to the left side of the access point of the current access sequence. Because the pages that replacement are always eliminated in memory, in this case, one of the three page numbers to chose. Consider taking the index of the access sequence point as the reference point to intercept - 1 size array on the left side of the access sequence, and then compare the three pages of memory allocation with the truncated array in turn. If the truncated sub array contains any, it will not be eliminated. Of course, it will only be compared three times. Take the one farthest from the left side of the current position in the three comparisons to exchange the new page in. During the implementation, it can take out the access sequence elements from the left side of the index point in turn. If the memory contains this element, the memory sequence will be progressively excluded; if the memory sequence does not contain this element or the memory sequence reaches the last element, the memory page will be eliminated. Since the data on the left side of the index point is considered every time, here call it "look left". According to this, the following core code Algorithm 2 can be extracted:
int index=0;               // Index of access sequence
int[] memList;             // Represents a sequence of stored memory.
int[] arrayList;           // Access sequence.
boolean lackPage;          // Indicates no page to visit and space for new pages.
boolean replacementPage;   // Indicates that the page needs to be eliminated.
for(int i;i<arrayList.length;i++){
    if(lackPage){
        add(newPage);     // Add new pages accessed in turn.
        index++;
        continue;
    }
    if(replacementPage){
        rep: for(index-1;index>=0;index--){
            for(int j=(memList.length-1);j>=0;j--){
                if(contains(arrayList[index])&&j!=0){
                    // Memory sequence exclude contained elements.
                    remove(memList[j]);
                    break;
                }
            }
        }
        pop(memList[index]);    // Eliminate the page number.
        add(newPage);      // New page add in memory.
        break rep;         // Break looking left.
    }
}

Algorithm 2. LRU algorithm's pseudo code and annotation.

Through verification, it gets that the number of page missing count in this example is 9 times. According to the formula of page missing rate \( P = \frac{\text{count}}{\text{length}} \times 100\% \), the rate of page missing is 75%. The result is consistent with the idea of LRU definition. However, it needs to be noted that this example is selected in a special way. So the conclusion compared with the page missing situation of FIFO algorithm, concluded that the number of page missing of LRU is less than that of FIFO. If changing different access sequences, it will get different page missing times and page missing rate. Of course, in the algorithm implementation, the initialization of index can not start from 0. In this case, it can start from 3, because the number of pages that can be stored in memory is 3 at the beginning, and page fault may occur from the beginning of the 4th access sequence. In addition, the logical judgment “Page-faults” could not be used. So this algorithm can be improved!! But here list them is to describe the unity among the three algorithms (FIFO, LRU, OPT).

4. OPT Algorithms
This is an ideal page replacement algorithm. Its basic idea is that when there is a page shortage, it can only be used for a long time in the future. This algorithm is practically impossible to implement! Because the access to the page is different for different processes, and it is unpredictable according to the characteristics of the system. When a page missing interrupt occurs, the OS cannot know when each page will be accessed next. But the idea of this algorithm is just the opposite of LRU algorithm. In addition, although this algorithm can't be implemented, one advantage of this algorithm is that it can be used to measure and compare the performance of the other algorithms. Because the number of page missing is the least and the most "ideal" in this algorithm.

Of course, studying this algorithm here, not to summarize the predecessors, but to consider the calculation of this algorithm. It can be considered that the idea of this algorithm is the opposite of that
of LRU algorithm. The implementation of computation can be considered according to this strategy (The physical implementation is really not possible!). Consider that in case of page fault, replace the pages that are the longest to be visited from the current access sequence index point in the memory sequence, and replace that one with the new page, which is the idea of this algorithm. Considering that LRU algorithm is the longest page replacement that has occurred from the current index point, it is the opposite of the ideal replacement algorithm. It can consider that the OPT algorithm is "look right". In this way, the three algorithms (FIFO, LRU, OPT) can be unified. In order to better describe the calculation of OPT algorithm, the same instance above is given. As shown in Table 4.

### Table 4. A OPT Example of right mode.

| No. | I  | II | III | IV | V  | VI | VII | VIII | IX | X  | XI | XII |
|-----|----|----|-----|----|----|----|-----|------|----|----|----|-----|
| As^a| 3  | 2  | 4   | 1  | 4  | 3  | 5   | 4    | 2  | 1  | 1  |     |
| Ma^b| 3  | 2  | 1   | 1  | 1  | 5  | 5   | 5    | 5  | 5  | 5  |     |
| Pf^c| √  | √  | √   | ×  | ×  | √  | ×   | √    | ×  | √  | ×  |     |

^a Access sequence.
^b Memory allocation.
^c Page fault. √ true, × false.

In Table 4, it can also divide the access sequence into three situations: 1st one is that there is no page needed in memory, and there is enough space for new pages to occupy; 2nd one is that there is no space for new pages, but the pages allocated in memory need to be eliminated and replaced with new ones; 3rd one is that there are pages in memory that need to be accessed without to do in and out operation. Of course, still focus on the realization of the second situation. But there is a special case in the second algorithm. When a page is eliminated, because the future page usage is considered, it will happen when a page is eliminated. There are many options, up to the total number minus one memory sequence to choose. Because only one certain page is eliminated, no matter which page is eliminated, these choices have no effect on the final quantitative evaluation value: The number of pages fault or page missing rate. In this case, for example, page 4 and page 3 can be eliminated at the X call, and one of all pages except page 5 can be eliminated at the XI invoking without affecting the page missing rate. But in any case, because the algorithm can not be realized in reality, it is only considered from calculation. Processing is to consider the right side of the index point. For this reason, it can still extract the following core code:
int index = 0; // Index of access sequence
int[] memList; // Represents a sequence of stored memory.
int[] arrayList; // Access sequence.
boolean lackPage; // Indicates no page to visit and space for new pages.
boolean replacmentPage; // Indicates that the page needs to be eliminated.
for (int i; i < arrayList.length; i++) {
    if (lackPage) {
        add(newPage); // Add new pages accessed in turn.
        index++;
        continue;
    }
    if (replacementPage) {
        rep: for (;
            index < arrayList.length; index++) {
            for (int j = (memList.length - 1); j >= 0; j--) {
                if (contains(arrayList[index]) && j != 0) {
                    // Memory sequence exclude contained elements.
                    remove(memList[j]);
                    break;
                } else {
                    pop(memList[j]); // Eliminate the page number.
                    add(newPage); // New page add in memory.
                    break rep; // Break looking right.
                }
            }
        }
    }
}

Algorithm 3. OPT algorithm's pseudo code and annotation.

Through verification, it could know that the number of page missing in this case is 7 times, and the page missing rate is 58.3%. It is the algorithm with the least number of page missing and the least rate of page missing. So it is also verified that this algorithm is a strategy to test the performance of other algorithms.

Also, it should be noted that the setting of “index” and “Page-faults” in the algorithm given is based on the idea of treating the three algorithms in a unified way. And through the above three algorithms can be unified into one "vertically and horizontally" idea.

5. Conclusion

This paper takes the dynamic page management of memory management in OS as the research aspect, and discusses three common page storage management algorithms: FIFO, LRU and OPT. Aiming at the idea that these three algorithms are not considered in a unified way, the algorithm idea of "looking up, down, left, right" is established. A small example is given to illustrate the common characteristics of the three algorithms. The memory access to pages can be divided into three situations, namely, the 1st situation in which pages need to be accessed, memory is not available, and memory has free space to store; the 2nd situation in which pages need to be accessed in the access sequence do not need to be replaced or eliminated in memory; the page number needs to be accessed is not in memory, and The physical block of memory allocation has occupied the space of the process, and a page needs to be eliminated to add a new page is the 3rd case. This paper discusses the three situations of these three algorithms (FIFO, LRU, OPT) respectively, and further describes the idea of unified view of the algorithm in the way of pseudo code.

In addition, for the example selected in this paper, the selection of access sequence is an example. In order to better reflect the uncertainty of page access in OS, 12 access sequences can be implemented by using the pseudo-random method between 1-9 to verify and simulate the experiment. Through these
many experiments, this still can achieve the calculation of these three algorithms, and ensure the accuracy of the conclusion. Of course, there are many descriptions of these three algorithms, but there is no unified way to consider these three algorithms, which is also the contribution of this paper.

References
[1] Andre S Tanenbam 2001 *Modern Operating System*(Second Edition)
[2] Abraham Silberschatz peter Baer Galivin Greg Gagne 2002 *Operating System Concepts*(Sixth Edition)
[3] Gary Nutt 2004 *Operation Systems*(Third Edition)
[4] Mhamed Zineddine 2018 *Optimizing security and quality of service in a real-time operating system using multi-objectionve bat algorithm* (Future Generation Computer Systems) vol 87 pp102-114
[5] Yousaf Bin Zikria *A survey on routing protocols supported by the Contiki Internet of things operation system* (Future Generation Computer Systems) vol 82 pp200-219
[6] Joseph Rahme Youssef *EOS:enterprise operation systems* (International Journal of Production Research)vol 56 pp2714-2732
[7] Peiquan Jin 2012 *AD-LRU:An efficient buffer replacement algorithm for flash-based database* (Data & Knowledge Engineering)vol 72 p83-102
[8] Marrcos Kiwi 2017 *FIFO Queues Are Bad for Rumor Spreading* (Information Theory, IEEE Transactions on)vol 63 pp1159-1166
[9] Maria Potop 2018 *FIFO Order reliable convergecast in WBAN* (Computer Networks)vol 146 pp200-216
[10] Maxim Finkelstein 2018 *On optimal replacement of systems with failure rates described by a random jump process* (Quality and Reliability Engineering International)vo 134 pp 1590-1604
[11] Mitra Folulandirad 2018 *Sensitivity of Optimal replacement policies to lifetime parameter estimates* (European Journal of Operational Research)vol 266 pp 963-975
[12] Maxim Finkelstein 2018 *On optimal replacement of systems with failure rates described by a random jump process* (Quality and Reliability Engineering International)vo 34 pp 1590-1604