2009 International Conference on Computer Engineering and Applications
IPCSIT vol.2 (2011) © (2011) IACSIT Press, Singapore

Reducing Page Faults with a Self Testing Page Allocation Policy

Omid Gholami[1], AmirMasoud Rahmani, Homayoun Motameni

Computer Engineering Department, Islamic Azad University
Mahmudabad Branch, Iran

Abstract. Virtual memory is a technique which allows the process to be run without being in the main memory completely due to lake of enough memory in system. Here techniques such as demand paging do the interchange of some parts of being executed process. Due to the slow work of secondary memory, this exchange of pages cost a lot and also slows down the system functionality. Different algorithms are designed for page replacement try to decrease page faults, each of which is valuable but an important point is the amount of allocated pages to the process which play an important role in decreasing the page faults. This paper tries to introduce a page allocation policy based on process behavior in different period of time which leads to the decrease of total page faults of the system and causes increase in system efficiency.

Keywords: Virtual Memory, Paging, Page Allocation Algorithm.

1. Introduction

With multi-tasking growth, the need to main memory is also increased. But generally this memory is not sufficient for all processes. So there is no way but to divide the space of the memory between processes. There fore each process is given less than what is needed. The process is saved in the secondary memory and is brought to the main memory to be executed. The **lazy exchange** technique is used instead of complete transition technique. It means that only parts of the process which are needed are taken into the main memory and are located in places considered for them [1].

Now if the process needs a new page which is not in the main memory operating system should remove one of pages from the memory and replace another page. There are different algorithms for page replacement of which we can name FIFO, LRC, Optimal, Second Chance, LFU and MFU [3]. All of these algorithms follow the same goal and that is to decrease the page fault and increase system efficiency and performance [3]. The important point here is the amount of pages allocated to each process, because more pages causes fewer page fault.

2. Related work

There are different algorithms such as **Equal Allocation** or **Appropriate Allocation** for allocating pages to process which will be discussed. The easiest method is allocation of **m** frame to **n** process to which **m/n** of frame is given to each process, for example if we have 93 frame and 5 process, 18 frame is given to each, and three are left. This method is called **Equal Allocation**.

Second method of frame allocation is to determine how many frames each process needs. If in one system with 62 free frames there are l0k and 127k process it is not good that each process is given 31 frames because a 10k process needs just 10 frames, and the rest (21 frames) are useless. In the second method of memory allocation, frames are given based on the processes size, which is called **Appropriate Allocation** [1]. Suppose that the required memory for process P_i is S_i, so the allocated memory to process is calculated as follows:

1 Omid Gholami, Tel: (+98)9111558339, Email: gholami_iran@yahoo.com

(1) $A_i = M*(S_i/S)$

In which A_i is the number of allocated page to process P_i, S_i is the required pages and S is equal to $\sum S_i$. M is total pages available. For example in the previous example the total number of required pages are 137 and is given to the process as mentioned here:

- (10 / 137) * 62 =4
- (127 / 137) * 62 =57

The third type of allocation is allocation based on Priority of the processes and it works in the way that each kind of process is given a priority and each priority has specific amount of pages [3].

3. Dynamic Page Allocation

The point which is not observed in above algorithms is degree of locality of requests for each process. In the previous example the first process is given 4 and the second on is given 57 frames. While it is possible that the first process has a lot of jumps to different parts of the program and has a higher page fault and the second process runs totally linear and has lower page fault. It is believed that in the number of allocated pages to one process, the amount of locality of required addresses is considered by process in time periods. It means that the process which accesses the memory regularly and has lower page fault is given less page and the process which has upper page fault is given more pages, So that totally the number of faults of total system pages are decreased and the performance is increased.

In this method the page fault is controlled periodically and according to it the amounts of the pages of the processes are calculated. For example maybe a process in a period of run has the high page fault but after a while the faults are decreased. In this case by increase of faults the numbers of pages are increased and by decrease of faults the numbers of pages are decreased [5].

To use this policy it is behaved in a way that at the beginning of the performance based on appropriate allocation, each process is given some frames and then the process behaviour is evaluated in a specific time period then the process is given new amounts of pages (the number of the pages of the process are decreased or increased). For example suppose that there are 3 processes (A, B and C) and also there are 30 frames of pages in the main memory. Based on appropriate allocation algorithm each one is given 10 frames (Suppose that the processes are the same size). After a specific time period the processes will be evaluated. If it is supposed that the number of faults of page A is 8 and B is 12 and C is 40, based on number of faults 30 frames are divided between them which are calculated as follows:

A: (8 / 60) × 30 = 4 frames

B: (12 / 60) × 30 = 6 frames

C: (40 / 60) × 30 = 20 frames

As it is seen process C which had the most page faults has held the most numbers of memory pages.

3.1. Implementation of Dynamic Allocation

To prove this idea simulation is used. In a way that 4 process are considered and are supposed to have the same size. They are divided into three levels based on locality of Addresses:

- ***Max*** is the processes which have few jumps and their required addresses are close to each other**.**
- ***Min*** is the processes with a lot of jumps and need page irregularly and their required addresses are totally expanded**.**
- ***Medium*** is the processes that based on the pages request are in the middle**.**

The simulator program at the beginning of the simulation divides the memory pages between the processes equally. For page replacement FIFO and Optimal algorithms were used. In this simulation in periods with seven requests of pages, the process history is used to find out how many page faults it had, then based on this statistics for next round, page decision will happen. The process with higher page fault will have more pages. It should be mentioned that in each period only the faults of prior period is considered to prevent wrong estimation. For example consider a process which had a lot of page fault at the beginning, now it does not have any request with page fault. If it is tired to allocate pages based on all faults the process will own a lot of pages which are not needed. So it is better to assess the processes periodically [6].

What was observed in this policy was not interesting. They also worked badly in some cases. In analyze it was shown that the algorithm has some weaknesses. It was observed when you considered the A process that does not have page fault in one simulation period. Now the number of this process page in the next period is zero ($(0\times60) / 30 = 0$).

This process has not taken any pages for next period. This problem causes the process faces a lot of page fault in the next run and it increases the total faults and decreases the algorithm efficiency. The above algorithm is directed this way so that the amount of page faults of all processes would be the same or to some extend balanced. To solve the pervious problem three constrain are considered for the algorithm:

First: In each level, algorithm can add or subtract one page to or form the number of pages of the process. So that the process which has a true function and has low page fault will not face severe page decrease and does not face the former algorithm problem.

Second: If the difference between the process page faults and the average of total process page faults is less that %20, the number of page will not change number of pages. Because when the number of process page fault is almost the same or close to each other, the cost of taking a page from one process and be given to another is more than the advantage of a page increase to a process.

Third: If the number of page of a process is less than a specific bound (limit) it should not be decreased again. Because the process which has few pages and has the most thrift in using the pages should not face any shortages (in simulation the least number of pages for a process is 3).

After enforcing these policies to simulator program a relative improvement was made in the number of all page faults in processes. Also surprisingly since the numbers of process page faults were almost the same there is a hope to divide power of a multi-tasking system between them equally. Because normally when program is faced a page fault there is an interrupt to bring the page into the memory. In the pervious situation the program which had a lot of page fault, lost a lot of time due to the interrupts and it led to a low performance comparing to a page faultless process. While with new policy it is enforced equally on all, so the programs will run more rightfully.

4. Evaluation

Four processes are considered for simulation. Also number of the existing pages in the memory can change between 30 and 60. The policy chosen for the pages replacement was **FIFO** and **Optimal**. Two processes are taken from MIN and two from Max level. Each program calls 20 pages from 1 to 20 (we can assume minimum fault for each process is 20 faults). Amounts of page request for each process are 300 pages. The results of simulation are in Table 1.

Table 1: Amounts of Page faults with new Page Allocation policy

Pages existing in system (total)	Total pages fault (4 process)		Page faults with *Self-Testing* Allocation		Page faults without *Self-Testing* Allocation	
	SELF-Testing FIFO	*Normal FIFO*	Most	Least	Most	Least
30	417	423	155	103	215	20
35	371	389	113	83	178	20
40	294	358	84	64	160	20
45	253	322	74	53	151	20
50	188	254	57	41	125	20
55	162	248	58	38	110	20
60	157	203	53	37	87	20
Pages existing in system (total)	Total pages fault (4 process)		Page faults with *Self-Testing* Allocation		Page faults without *Self-Testing* Allocation	
	SELF-Testing Optimal	*Normal Optimal*	Most	Least	Most	Least
30	411	421	152	103	213	20
35	262	380	105	80	170	20
40	266	332	78	59	149	20
45	239	301	66	43	139	20
50	176	242	57	39	117	20
55	159	233	54	37	103	20
60	158	206	49	36	80	20

Graph in Fig. 1 shows the number of faults for range between 30 and 70 frames for page allocation without and with past page fault evaluation and with FIFO page replacement policy.

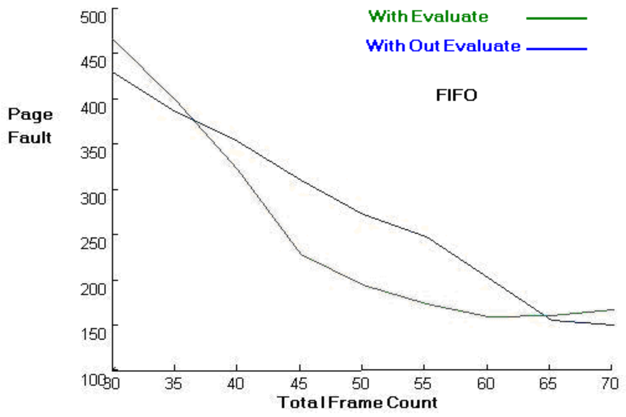

Fig. 1: Number of page faults with and without Evaluation (FIFO page replacement policy)

Graph in Fig. 2 shows the number of faults for range between 30 and 70 frames for page allocation without and with past page evaluation and with Optimal page replacement policy.

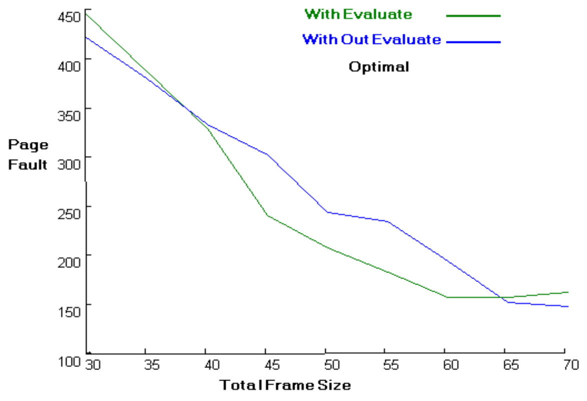

Fig. 2: Number of page faults with and without Evaluation (Optimal page replacement policy)

As it is observed in range of 45 to 60 frame evaluation policy could show good efficiency and high difference in fault amount with normal way; while by increase in number of frames the efficiency will decrease because increase in number of frames will be helpful to all of process and algorithm does not have much to do [7]. Also when number of pages are too low this policy is not helpful, because the faults will influence all processes and you can not find a process with low page faults to allocate its extra pages to others. The other point is that the least page fault for processes is 20, because it is supposed that the processes have the same size and they all need 20 page frames. So they will have page fault for first request of the frames.

The only thing that must be mentioned is that this policy makes CPU Usage overload. Measurements showed that CPU usage without this policy in 10 evaluations was about 35% for simulator and with this Policy it became about 46%.

5. Conclusion

As it was observed the page allocation base on past page faults of processes presents a good result totally in a multi-tasking systems and it decreased the number of system page faults. Also by creating balance in number of faults of pages in difference programs it helps in better functionality of operating system. For future work we could contemplate other parameter like time or priority of processes in page allocation.

6. References

[1] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, *"Operating System Concept"*, 7Th Edition, 2004.

[2] Alvin R. Lebeck, Xiaobo Fan, Heng Zeng, Carla Ellis, *"Power Aware Page Allocation"*, Department of Computer Science, Duke University, Durham, NC 27708 USA, falvy,xiaobo,zengh,carlag@cs.duke.edu, 2000.

[3] Ann Arber, *"Virtual Memory Management"*, MI: UMAI Research Press, 1984.

[4] Murali Vilayannur, Anand Sivasubramaniam, Mahmut Kandemir, *"Pro-active Page Replacement for Scientific Applications: A Characterization"*, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802,USA,{vilayann,anand,kandemir}@cse.psu.edu, 2005 IEEE.

[5] Yannis Smaragdakis, Scott Kaplan, Paul Wilson, *"The EELRU adaptive replacement algorithm",* Georgia Institute of Technology, Atlanta, GA 30332-0155, USA, October 2002.

[6] Susanne Albersa, Lene M. Favrholdtb, Oliver Gielc, *"Paging with locality of reference"*, Department of Mathematics and Computer Science, University of Southern Denmark, 2004.

[7] Nimrod Megiddo, Dharmendra S. Modha, *"ARC: A SELF-TUNING, LOWOVERHEAD REPLACEMENT CACHE"*, USENIX File & Storage Technologies Conference (FAST), San Francisco, CA, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, March 31, 2003.