TP n°5: Management memory (Part 2)

Page replacement algorithms

1. Objectives

This practical work has the following objectives:

- Understand how page replacement algorithms work in the event of a page fault,
- Familiarize yourself with the programming environment for page replacement algorithms,
- Know how to measure the performance of page replacement algorithms.

2. Introduction to page faults and their impact

2.1 Page fault

Page faults are common in computer systems, and understanding their impact is crucial to optimizing performance. In simple terms, a page fault occurs when a program attempts to access a page of memory that is not currently present in the system's physical memory (RAM). This causes the operating system to retrieve the required page from secondary storage (for example, the hard disk) into main memory (RAM), allowing the program to continue running.

2.2 The impact of page faults

Although page faults are an essential mechanism for managing the memory of a computer system, they can have a significant impact on overall performance. When a page fault occurs, the processor is forced to wait for the required page to be retrieved from secondary storage, resulting in a delay in program execution. This delay can be particularly noticeable when large datasets or memory-intensive applications are involved.

To better illustrate the impact of page faults, let's consider a scenario in which photo editing software is running on a computer with limited physical memory. When the user opens a large image file, the software may need to access several pages of memory that are not currently present. This triggers a series of page faults, preventing the software from responding until the required pages are retrieved from secondary storage. The user experiences a delay in their workflow, reducing productivity and satisfaction.

2.3 Strategies for reducing page faults

There are strategies and techniques that can be used to alleviate the impact of page faults and optimize system performance. Here are some strategies to consider:

a) Increase physical memory: one of the most effective ways of reducing page faults is to increase the amount of physical memory available in the system. With more memory, the probability of pages being present when needed is considerably higher, which reduces the occurrence of page faults.

b) Use high-performance replacement algorithms to reduce the number of page faults.

3. Page replacement algorithms

Page replacement algorithm is used to decide which page will be replaced to allocate memory to the current referenced page. Different page replacement algorithms suggest different ways to decide which page is to be replaced. The main objective of these algorithms is to reduce the number of page faults.

A page replacement algorithm is designed to minimize the number of page faults. We are looking for the algorithm that best reduces the probability of a page fault occurring.

An algorithm is evaluated by taking a string of page numbers and counting the number of page faults that occur during this sequence of accesses, as a function of the number of pages of main memory available to it.

To illustrate the replacement algorithms, we will use the following sequence of pages:

7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1 and 3 pages (frames) in main memory.

3.1 Optimum replacement

Use as a target the page that will not be used for the longest time.

So for our suite: `7xx 70x 701 201 - 203 - 243 - -203 - - 201 - - - 701 - -`

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
p0	**7**	7	7	**2**		2		2			2			2				**7**		
p1		**0**	0	0		0		**4**			**0**			0				0		
p2			**1**	1		**3**		3			3			**1**				1		
Page fault	x	x	x	x		x		x			x			x				x		

3.2 FIFO replacement

The simplest algorithm is First-In-First-Out. When a target is to be selected, the oldest page is selected.

In this algorithm, the S.E keeps track of all the pages in memory in a queue, with the oldest page at the head of the queue. When a page needs replacing, the page at the top of the queue is selected for deletion.

So for our suite: `7XX/70X/701/201-201/231/230/430/420/423/023-023-023/013/012-012-012/712/702/701`

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
p0	**7**	7	7	**2**		2	2	**4**	4	4	**0**			0	0			**7**	7	7
p1		**0**	0	0		**3**	3	3	**2**	2	**2**			**1**	1			1	**0**	0
p2			**1**	1		1	**0**	0	0	**3**	3			3	**2**			2	2	**1**
Page fault	x	x	x	x		x	x	x	x	x	x			x	x			x	x	x

That is, fifteen page faults.

Unfortunately, this quick and easy-to-program mechanism is not very effective. There are sequences of pages for which this algorithm produces more page faults with four memory pages than with three! (for example: 1,2,3,4,1,2,5,1,2,3,4,5). This is the Belady anomaly.

3.3 LRU (Least Recently Used page)

Here we use the ageing of a page rather than the order in which the page was created. We are betting that pages that have been used recently will be used in the near future, while pages that have not been used for a long time are no longer useful.

So much for our suite: `7xx 70x 701 201 - 203 - 403 402 432 032 - - 132 - 102 - 107 -`

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
p0	**7**	7	7	**2**		2		**4**	4	4	**0**			**1**		1		1		
p1		**0**	0	0		0		**0**	0	**3**	3			3		**0**		0		
p2			**1**	1		**3**		3	**2**	2	2			2		2		**7**		
Page fault	x	x	x	x		x		x	x	x	x			x		x		x		

That is, 12-page faults.

The LRU algorithm is a good one, but it poses numerous implementation problems and may require substantial hardware tools.

Software solutions:

a) Counters: A time counter is added to each entry in the table of pages and is updated each time the page is accessed. You need to search the entire table for the victim. In addition, these times must be updated when the page table is changed (to that of another process, etc.). Real time cannot be used...

b) A stack: Each time a page is accessed, the page is placed at the top of the stack. The top of the stack is always the most recently used page and the bottom of the stack the least recently used page.

c) Masks: A byte is associated with each page. The system sets the most significant bit to 1 each time the page is accessed. Every N milliseconds (clock click, N = 100) the system shifts the byte associated with each page to the right. This provides a history of page usage. The byte at 00000000 indicates that the page has not been used for 8 cycles, 11111111 indicates that the page has been used for 8 cycles. The mask page 11000100 has been used more recently than 01110111. If we interpret these bytes as unsigned integers, it is the page with the smallest byte that has been used least recently (as the uniqueness of the numbers is not guaranteed, the selection between identical numbers is made using the FIFO order).

3.4 The second chance algorithm

A bit associated with each page is set to 1 each time a page is used by a process. Before removing a page from memory, we try to give it a second chance. We use a FIFO plus second chance algorithm:

If the use bit is 0, the page is swapped out of memory (it has not been used since the last page request). If the bit is 1, it is set to zero and another victim is sought. This page will only be swapped out of memory if all the other pages have been used, and are also using their second chance.

You can think of this as a circular queue, where you swap pages that have the bit set to 1 (by setting it to zero) until you find a page with the use bit set to zero.

3.5 Most frequently used (MFU)

As its name indicates, it is frequency of use that comes into play instead of age, but it is the same mechanism as LRU. These two algorithms, LRU and MFU, are rarely used because they require too much computing time and are difficult to implement, but they are quite effective.

4. Performance criteria of page replacement algorithms

The performance of a page replacement allocation algorithm can be measured according to criteria such as:

- *Number of page faults*
- *Page fault ratio*: the ratio between the number of page faults and the total number of pages referenced.

Exercises

Exercise n°1

To understand how page replacement algorithms work, you will be asked to implement the following algorithms:

- FiFO
- LRU
- Optimal
- Second chance

In order to carry out this simulation, the following points must be taken into account:

- Read the reference string (sequence of page numbers).

- Specify the memory size and page size, then calculate the number of memory cells (frames).

- Choose the replacement algorithm and give the number of page faults, specifying the different states of memory.

- Write the corresponding program for each algorithm, choosing the necessary data structures.

- Apply to several examples.