

EOPSY Lab #4

Baran Korkmaz 302809

Introduction

The aim of this exercise is to simulate and understand how Operating Systems deal with page faults and to observe the execution of the page replacement by using a memory management simulator.

Theoretical Information

Physical Memory refers to the actual RAM of the system, which usually takes the form of cards attached onto the motherboard. Also called primary memory, it is the only storage type directly accessible to the CPU and holds the instructions of programs to execute.

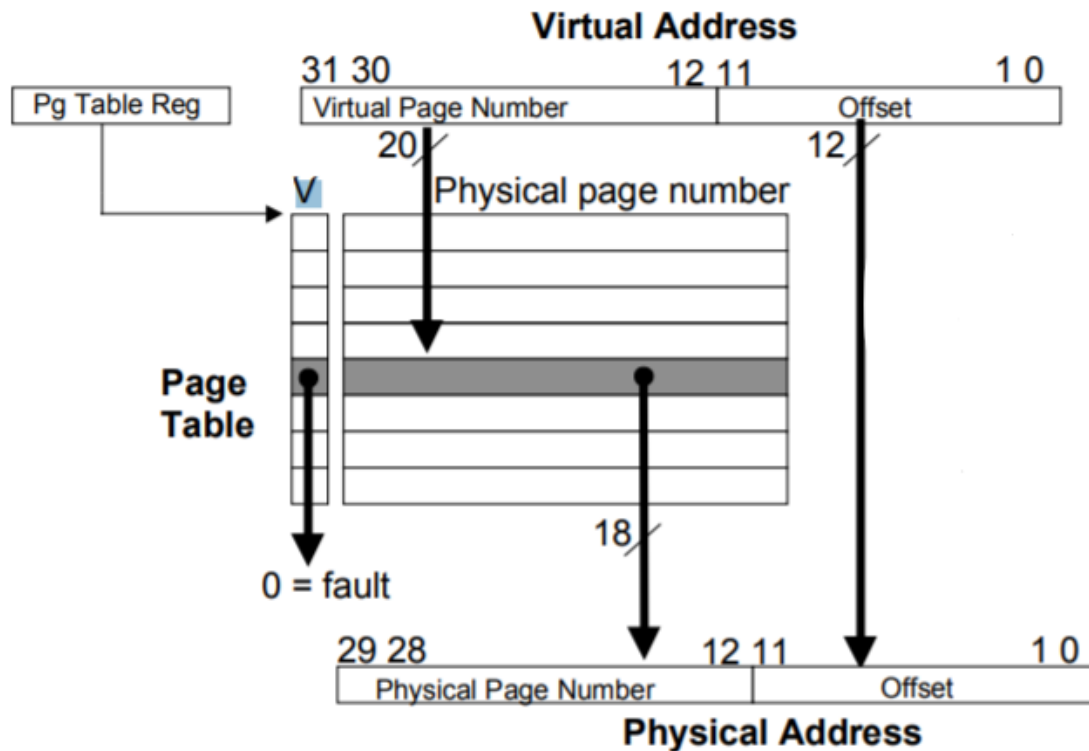
Virtual Memory is the process of mapping a logical address space numbered from 0 to the physical address space of the computer so that the RAM serves as a cache for the program's memory stored on the disk. It maps memory addresses used by a program, called **virtual addresses**, into **physical addresses** in computer memory. Virtual addresses, also called logical addresses, are program generated. In a system without virtual memory, a physical address and a logical address are the same; the virtual address is placed on the address bus directly because it refers to a physical address in the main memory. On systems with virtual memory, there is a mapping between virtual addresses and real addresses.

Virtual memory solves several critical problems:

- Simplifying RAM memory management by separating the address of memory from its physical location
- Providing protection for users by giving them their own address spaces
- Simplifying compilation and usage of libraries by allowing all programs to begin addressing memory from zero

The idea behind virtual memory is that physical memory is divided into **fixed-size pages**. Loadable modules are also divided into several **page frames**. Page frames are always the same size as the pages in memory.

Pages frames are loaded into memory only when they are needed. Adjacent page frames are not necessarily loaded into adjacent pages in memory. At a point in time during the execution of a program, only a small fraction of the total pages of a process may be loaded.



In **Demand paging**, only a set of pages of a process is loaded into the memory. This is done so that we can have more processes in the memory at the same time.

When a page that is residing in virtual memory is requested by a process for its execution, the Operating System needs to decide which page will be replaced by this requested page. This process is known as page replacement and is a vital component in virtual memory management.

A **page fault** is an interruption that occurs when a software program attempts to access a memory block not currently stored in the system's RAM. This exception tells the operating system to find the block in virtual memory so, it can be sent from a device's storage to RAM. Once all the data has been placed into physical memory, the program resumes normal operation. The Page fault process occurs in the background, and thus the user is unaware of it.

When a page fault occurs, the Operating System will have to replace one of the existing pages with the newly needed page. There are different **page replacing algorithms** exist with all aim to reduce the number of page faults.

Page Replacement Algorithms

Algorithms used to page replacement by the operating system to deal with page faults.

First in First Out Algorithm

In this algorithm, a queue of all the pages that are in the memory is currently maintained. The oldest page in memory is at the front of the queue and the most recent page is at the back of the queue.

Whenever a page fault occurs, the operating system looks at the front of the queue to get the page to be replaced by the newly requested page. It also adds this newly requested page at the end and removes the oldest page from the front of the queue.

Least Recently Used Algorithm

This algorithm keeps track of usage of pages over a period. This algorithm works based on the principle of locality of a reference which states that a program tends to access the same set of memory locations repetitively over a brief period. So, pages that have been used heavily in the past are most likely to be used heavily in the future also.

When a page fault occurs, then the page that has not been used for the longest duration of time is replaced by the newly requested page.

Last In First Out Algorithm

In this algorithm, the newest page is replaced by the requested page. Usually, this is done through a stack, where we maintain a stack of pages currently in the memory with the newest page being at the top.

When a page fault occurs, the page at the top of the stack is replaced.

Random Page Replacement

This method is choosing any random page in the memory to be replaced by the requested page.

Optimal Page Replacement

In this algorithm, the pages are replaced with the ones that will not be used for the longest duration of time in the future. In simple terms, the pages that will be referred to farthest in the future are replaced in this algorithm.

Procedure

Setup

First, we needed to map the first 8 virtual pages to any 8 physical pages. To achieve this, memory.conf file updated as:

```
1 memset 0 1 0 0 0 0
2 memset 1 3 0 0 0 0
3 memset 2 2 0 0 0 0
4 memset 3 4 0 0 0 0
5 memset 4 6 0 0 0 0
6 memset 5 0 0 0 0 0
7 memset 6 7 0 0 0 0
8 memset 7 5 0 0 0 0
9
10 enable_logging true
11
12 log_file tracefile
13
14 pagesize 128
15
16 addressradix 10
17
18 numpages 64
```

We can also observe that page size assigned as 128 Bytes and address radix as 10 so it will behave like decimal numbers.

Second, we needed to configure the simulator for reading once from 64 pages. We achieved that by editing the command file as:

```
READ 1
READ 129
READ 257
.
.
READ 7681
READ 7809
READ 7937
```

There are 64 read statements (one for each page) and numbers after each indicate the page address. We start at 1 and since every page has a 128-byte size, we shift 128 bytes to get to the next page.

Simulation

The initial state of the simulator is shown below. It is expected that the first 8 pages of virtual memory are mapped to 8 pages as configured before.

run	step	reset	exit	status: STOP
virtual	physical	virtual	physical	time: 0
page 0	page 5	page 32		
page 1	page 0	page 33		instruction: NONE
page 2	page 2	page 34		address: NULL
page 3	page 1	page 35		
page 4	page 3	page 36		page fault: NO
page 5	page 7	page 37		
page 6	page 4	page 38		virtual page: x
page 7	page 6	page 39		physical page: 0
page 8	page 8	page 40		R: 0
page 9	page 9	page 41		M: 0
page 10	page 10	page 42		inMemTime: 0
page 11	page 11	page 43		lastTouchTime: 0
page 12	page 12	page 44		low: 0
page 13	page 13	page 45		high: 0
page 14	page 14	page 46		
page 15	page 15	page 47		
page 16	page 16	page 48		
page 17	page 17	page 49		
page 18	page 18	page 50		
page 19	page 19	page 51		
page 20	page 20	page 52		
page 21	page 21	page 53		
page 22	page 22	page 54		
page 23	page 23	page 55		
page 24	page 24	page 56		
page 25	page 25	page 57		
page 26	page 26	page 58		
page 27	page 27	page 59		
page 28	page 28	page 60		
page 29	page 29	page 61		
page 30	page 30	page 62		
page 31	page 31	page 63		

After stepping through one by one in the first 31 pages, there was no page fault observed. This is because the program has not tried to access a memory block not stored in the physical memory.

We can see from the below figure at the 33rd virtual page we have a flag for page fault.

run	step	reset	exit	status: STOP
virtual	physical	virtual	physical	time: 330 (ns)
page 0		page 32	page 1	
page 1	page 0	page 33		instruction: READ
page 2	page 2	page 34		address: 4097
page 3	page 1	page 35		
page 4	page 3	page 36		page fault: YES
page 5	page 7	page 37		
page 6	page 4	page 38		virtual page: 32
page 7	page 6	page 39		physical page: -1
page 8	page 8	page 40		R: 0
page 9	page 9	page 41		M: 0
page 10	page 10	page 42		inMemTime: 0
page 11	page 11	page 43		lastTouchTime: 0
page 12	page 12	page 44		low: 4096
page 13	page 13	page 45		high: 4223
page 14	page 14	page 46		
page 15	page 15	page 47		
page 16	page 16	page 48		
page 17	page 17	page 49		
page 18	page 18	page 50		
page 19	page 19	page 51		
page 20	page 20	page 52		
page 21	page 21	page 53		
page 22	page 22	page 54		
page 23	page 23	page 55		
page 24	page 24	page 56		
page 25	page 25	page 57		
page 26	page 26	page 58		
page 27	page 27	page 59		
page 28	page 28	page 60		
page 29	page 29	page 61		
page 30	page 30	page 62		
page 31	page 31	page 63		

At this point, the simulator replaced the first page in the queue to newly requested page. This gives us a clue that this simulator might be using First In First Out algorithm.

Once we continue with the simulation, it is clear that, this simulator indeed uses the First In First Out algorithm since it continues to replace it from the first page from the queue.

virtual	physical	virtual	physical	time:	460 (ns)
page 0		page 32	page 1		
page 1		page 33	page 3	instruction:	READ
page 2		page 34	page 2	address:	5761
page 3		page 35	page 4		
page 4		page 36	page 6	page fault:	YES
page 5		page 37	page 0		
page 6		page 38	page 7	virtual page:	45
page 7		page 39	page 5	physical page:	-1
page 8		page 40	page 8	R:	0
page 9		page 41	page 9	M:	0
page 10		page 42	page 10	inMemTime:	0
page 11		page 43	page 11	lastTouchTime:	0
page 12		page 44	page 12	low:	5760
page 13		page 45	page 13	high:	5887
page 14	page 14	page 46			
page 15	page 15	page 47			
page 16	page 16	page 48			
page 17	page 17	page 49			
page 18	page 18	page 50			
page 19	page 19	page 51			
page 20	page 20	page 52			
page 21	page 21	page 53			
page 22	page 22	page 54			
page 23	page 23	page 55			
page 24	page 24	page 56			
page 25	page 25	page 57			
page 26	page 26	page 58			
page 27	page 27	page 59			
page 28	page 28	page 60			
page 29	page 29	page 61			
page 30	page 30	page 62			
page 31	page 31	page 63			

The same outcome can be observed from the trace file:

1 READ 1 ... okay	34 READ 4225 ... page fault
2 READ 129 ... okay	35 READ 4353 ... page fault
3 READ 257 ... okay	36 READ 4481 ... page fault
4 READ 385 ... okay	37 READ 4609 ... page fault
5 READ 513 ... okay	38 READ 4737 ... page fault
6 READ 641 ... okay	39 READ 4865 ... page fault
7 READ 769 ... okay	40 READ 4993 ... page fault
8 READ 897 ... okay	41 READ 5121 ... page fault
9 READ 1025 ... okay	42 READ 5249 ... page fault
10 READ 1153 ... okay	43 READ 5377 ... page fault
11 READ 1281 ... okay	44 READ 5505 ... page fault
12 READ 1409 ... okay	45 READ 5633 ... page fault
13 READ 1537 ... okay	46 READ 5761 ... page fault
14 READ 1665 ... okay	47 READ 5889 ... page fault
15 READ 1793 ... okay	48 READ 6017 ... page fault
16 READ 1921 ... okay	49 READ 6145 ... page fault
17 READ 2049 ... okay	50 READ 6273 ... page fault
18 READ 2177 ... okay	51 READ 6401 ... page fault
19 READ 2305 ... okay	52 READ 6529 ... page fault
20 READ 2433 ... okay	53 READ 6657 ... page fault
21 READ 2561 ... okay	54 READ 6785 ... page fault
22 READ 2689 ... okay	55 READ 6913 ... page fault
23 READ 2817 ... okay	56 READ 7041 ... page fault
24 READ 2945 ... okay	57 READ 7169 ... page fault
25 READ 3073 ... okay	58 READ 7297 ... page fault
26 READ 3201 ... okay	59 READ 7425 ... page fault
27 READ 3329 ... okay	60 READ 7553 ... page fault
28 READ 3457 ... okay	61 READ 7681 ... page fault
29 READ 3585 ... okay	62 READ 7809 ... page fault
30 READ 3713 ... okay	63 READ 7937 ... page fault
31 READ 3841 ... okay	
32 READ 3969 ... okay	
33 READ 4097 ... page fault	

Conclusions

- The First In First Out algorithm was used in this simulation. It is a simple algorithm that is not optimal since it does not make a difference between pages frequently used or not used at all like other algorithms such as Least Recently Used or Optimal Page Replacement.
- This simulator creates an equal number of physical and virtual memory pages and since we configured it to have 64 total pages, it created 32 virtual and 32 physical memory pages.
- Since it creates 32 virtual memory pages, but we needed to read from 64, page faults started occurring on the 33rd page. The simulator then used the replacement algorithm to find a block in virtual memory and map it to physical memory.