# EOPSY LAB 4: Memory management

## Shailesh Kumar

## Introduction

A page fault occurs when a program attempts to access a block of memory that is not stored in the physical memory, or RAM. The fault notifies the operating system that it must locate the data in virtual memory, then transfer it from the storage device, such as an HDD or SSD, to the system RAM.

Virtual to physical address translation is realized with a set of a concept called Page Table. Such a table contains piece of information where the address (page) lies in RAM. In a situation, in which program requests a page from memory, and the page lies on the storage device, then a page fault is generated. Operating system has to handle it. While doing it, it chooses a page that is being removed from RAM and places needed page onto free place. Afterwards, Page Table is updated.

## Virtual memory

If a computer runs out of physical memory it can map some portion of it into secondary memory (such as external storage devices). This part of the memory acts as if it was a part of the main memory and is called virtual memory. The implementation can be done via either paging or segmentation.

## Paging

Paging consists in dividing virtual memory into blocks of identical size called pages, dividing the main memory into page frames and transferring these between virtual and physical memory. This is done with the use of a page table which contains the mapping between virtual and physical addresses and contains a frame table which keeps the information on which frames are mapped.

## Page replacement and page fault

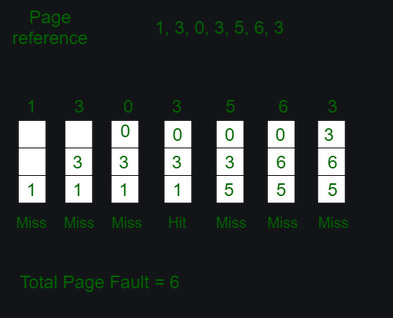
If, after checking the page table, we find out that the page we want to reference is not present in physical memory a “page fault” exception is raised and we need to decide which page to either page (or swap) in and which to page (or swap) out. This is done via one of many page replacement algorithms.

## Page replacement algorithms

Choosing the page to remove from RAM is done page replacement algorithms

1. FIFO – First in First out algorithm, this is the algorithm used in the code as well and you can have a look in pagefault.java

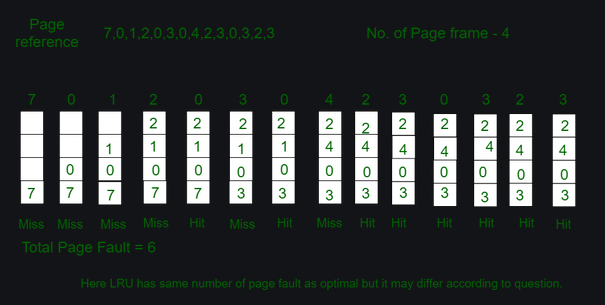
In event of a page fault and the page table being full, we replace the oldest entry/frame from the page table and replace it with the newest. Hence the name, FIFO



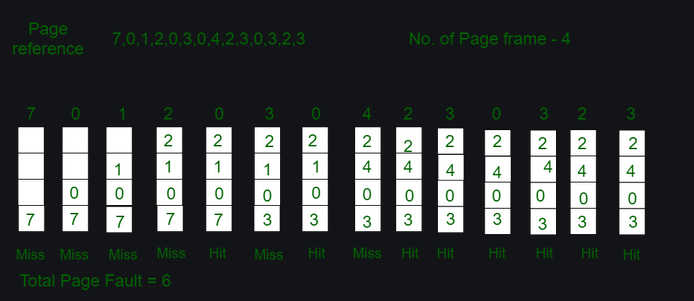
We see that 1 is replaced first and then 3 and then 0. (FIFO)

2. Least recently used. - In this one, the page is replaced which was used least recently. If you had 3 frames available and the instructions were 7 0 7 2.

After the 3 misses, for 2, 0 would be replaced from the page table.

We can see that 1 is replaced when 4 comes along, but FIFO, 0 would be the one that is replaced.

3. Optimal page replacement – this is just theoritical and used to measure other replacement algorithm. It is based on future instructions to OS, so it would replace a page which won’t be seen for the longest time.



We can see that first 7 is replaced since there is no more 7 after the initial one. Similary, 1 is replaced to give place to 4. This is theoritical since it is based on future instructions.

Reference for the theory - <https://www.geeksforgeeks.org/page-replacement-algorithms-in-operating-systems/>

How long does page fault take - <https://www.youtube.com/watch?v=bShqyf-hDfg&ab_channel=DavidBlack-Schaffer>

Udacity page fault short video - <https://www.youtube.com/watch?v=uqQOIahM868&ab_channel=Udacity>

## Task

This task requires 8 physical pages to be mapped to first 8 pages of virtual memory. Software solution provided with the tasks contains a bug – pages 8 - 31 are also mapped onto virtual memory ones (because read trial ends up with success). Page size is 4KB so we can access paged lying under addresses up to 0x8000. Otherwise one receives page fault exception – no physical pages are bonded with these virtual addresses.

Pages are assigned with a use of FIFO algorithm. We can conclude that because while encountering page faults, pages were replaced sequentially in the same order they were bonded with initial pages one had to map (this can be seen by clicking at pages’ buttons in the simulator and reading underlying addresses).

Commands:

We just write to the first 8 addresses and then Read all addresses in sequence.

We get a page fault after the first 32 read calls, since the page table is full. And each subsequent call requires a swap in/out. Since the algorithm used is FIFO, we can see that the frames replaced are in sequence.

WRITE hex 0

WRITE hex 4000

WRITE hex 8000

WRITE hex c000

WRITE hex 10000

WRITE hex 14000

WRITE hex 18000

WRITE hex 1c000

READ 0

READ 16384

READ 32768

READ 49152

READ 81920

READ 98304

READ 65536

READ 114688

READ 131072

READ 147456

READ 163840

READ 180224

READ 196608

READ 212992

READ 229376

READ 245760

READ 262144

READ 278528

READ 294912

READ 311296

READ 327680

READ 344064

READ 360448

READ 376832

READ 393216

READ 409600

READ 425984

READ 442368

READ 458752

READ 475136

READ 491520

READ 507904

READ 524288

READ 540672

READ 557056

READ 573440

READ 589824

READ 606208

READ 622592

READ 638976

READ 655360

READ 671744

READ 688128

READ 704512

READ 720896

READ 737280

READ 753664

READ 770048

READ 786432

READ 802816

READ 819200

READ 835584

READ 851968

READ 868352

READ 884736

READ 901120

READ 917504

READ 933888

READ 950272

READ 966656

READ 983040

READ 999424

READ 1015808

READ 1032192

Memory.conf:

// memset virt page # physical page # R (read from) M (modified) inMemTime (ns) lastTouchTime (ns)

memset 0 2 0 0 0 0

memset 1 20 0 0 0 0

memset 2 4 0 0 0 0

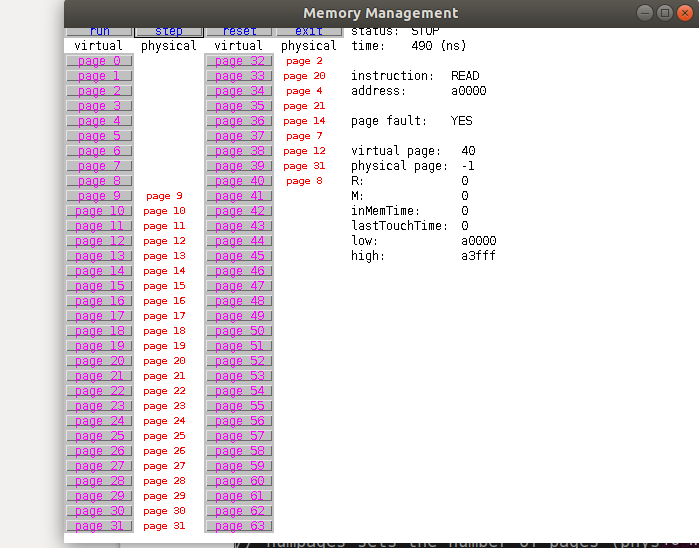
memset 3 21 0 0 0 0

memset 4 14 0 0 0 0

memset 5 7 0 0 0 0

memset 6 12 0 0 0 0

memset 7 31 0 0 0 0



We can see that when page fault does occur, the pages replaced are first in first out in sequential order

(2,20,4,21,14,7,12,31,8) just like from memory.conf

Tracefile:

WRITE 0 ... okay

WRITE 4000 ... okay

WRITE 8000 ... okay

WRITE c000 ... okay

WRITE 10000 ... okay

WRITE 14000 ... okay

WRITE 18000 ... okay

WRITE 1c000 ... okay

READ 0 ... okay

READ 4000 ... okay

READ 8000 ... okay

READ c000 ... okay

READ 14000 ... okay

READ 18000 ... okay

READ 10000 ... okay

READ 1c000 ... okay

READ 20000 ... okay

READ 24000 ... okay

READ 28000 ... okay

READ 2c000 ... okay

READ 30000 ... okay

READ 34000 ... okay

READ 38000 ... okay

READ 3c000 ... okay

READ 40000 ... okay

READ 44000 ... okay

READ 48000 ... okay

READ 4c000 ... okay

READ 50000 ... okay

READ 54000 ... okay

READ 58000 ... okay

READ 5c000 ... okay

READ 60000 ... okay

READ 64000 ... okay

READ 68000 ... okay

READ 6c000 ... okay

READ 70000 ... okay

READ 74000 ... okay

READ 78000 ... okay

READ 7c000 ... okay

READ 80000 ... page fault

READ 84000 ... page fault

READ 88000 ... page fault

READ 8c000 ... page fault

READ 90000 ... page fault

READ 94000 ... page fault

READ 98000 ... page fault

READ 9c000 ... page fault

READ a0000 ... page fault

READ a4000 ... page fault

READ a8000 ... page fault

READ ac000 ... page fault

READ b0000 ... page fault

READ b4000 ... page fault

READ b8000 ... page fault

READ bc000 ... page fault

READ c0000 ... page fault

READ c4000 ... page fault

READ c8000 ... page fault

READ cc000 ... page fault

READ d0000 ... page fault

READ d4000 ... page fault

READ d8000 ... page fault

READ dc000 ... page fault

READ e0000 ... page fault

READ e4000 ... page fault

READ e8000 ... page fault

READ ec000 ... page fault

READ f0000 ... page fault

READ f4000 ... page fault

READ f8000 ... page fault

READ fc000 ... page fault