# 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

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.

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.

## 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:

// Enter READ/WRITE commands into this file

// READ <OPTIONAL number type: bin/hex/oct> <virtual memory address or random>

// WRITE <OPTIONAL number type: bin/hex/oct> <virtual memory address or random>

READ hex 1

READ hex 4001

READ hex 8001

READ hex C001

READ hex 10001

READ hex 14001

READ hex 18001

READ hex 1C001

READ hex 20001

READ hex 24001

READ hex 28001

READ hex 2C001

READ hex 30001

READ hex 34001

READ hex 38001

READ hex 3C001

READ hex 40001

READ hex 44001

READ hex 48001

READ hex 4C001

READ hex 50001

READ hex 54001

READ hex 58001

READ hex 5C001

READ hex 60001

READ hex 64001

READ hex 68001

READ hex 6C001

READ hex 70001

READ hex 74001

READ hex 78001

READ hex 7C001

READ hex 80001

READ hex 84001

READ hex 88001

READ hex 8C001

READ hex 90001

READ hex 94001

READ hex 98001

READ hex 9C001

READ hex A0001

READ hex A4001

READ hex A8001

READ hex AC001

READ hex B0001

READ hex B4001

READ hex B8001

READ hex BC001

READ hex C0001

READ hex C4001

READ hex C8001

READ hex CC001

READ hex D0001

READ hex D4001

READ hex D8001

READ hex DC001

READ hex E0001

READ hex E4001

READ hex E8001

READ hex EC001

READ hex F0001

READ hex F4001

READ hex F8001

READ hex FC000

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

// enable\_logging 'true' or 'false'

// When true specify a log\_file or leave blank for stdout

enable\_logging true

// log\_file <FILENAME>

// Where <FILENAME> is the name of the file you want output

// to be print to.

log\_file tracefile

// page size, defaults to 2^14 and cannot be greater than 2^26

// pagesize <single page size (base 10)> or <'power' num (base 2)>

pagesize 16384

// addressradix sets the radix in which numerical values are displayed

// 2 is the default value

// addressradix <radix>

addressradix 16

// numpages sets the number of pages (physical and virtual)

// 64 is the default value

// numpages must be at least 2 and no more than 64

// numpages <num>

numpages 64

tracefile:

READ 1 ... okay

READ 4001 ... okay

READ 8001 ... okay

READ c001 ... okay

READ 10001 ... okay

READ 14001 ... okay

READ 18001 ... okay

READ 1c001 ... okay

READ 20001 ... okay

READ 24001 ... okay

READ 28001 ... okay

READ 2c001 ... okay

READ 30001 ... okay

READ 34001 ... okay

READ 38001 ... okay

READ 3c001 ... okay

READ 40001 ... okay

READ 44001 ... okay

READ 48001 ... okay

READ 4c001 ... okay

READ 50001 ... okay

READ 54001 ... okay

READ 58001 ... okay

READ 5c001 ... okay

READ 60001 ... okay

READ 64001 ... okay

READ 68001 ... okay

READ 6c001 ... okay

READ 70001 ... okay

READ 74001 ... okay

READ 78001 ... okay

READ 7c001 ... okay

READ 80001 ... page fault

READ 84001 ... page fault

READ 88001 ... page fault

READ 8c001 ... page fault

READ 90001 ... page fault

READ 94001 ... page fault

READ 98001 ... page fault

READ 9c001 ... page fault

READ a0001 ... page fault

READ a4001 ... page fault

READ a8001 ... page fault

READ ac001 ... page fault

READ b0001 ... page fault

READ b4001 ... page fault

READ b8001 ... page fault

READ bc001 ... page fault

READ c0001 ... page fault

READ c4001 ... page fault

READ c8001 ... page fault

READ cc001 ... page fault

READ d0001 ... page fault

READ d4001 ... page fault

READ d8001 ... page fault

READ dc001 ... page fault

READ e0001 ... page fault

READ e4001 ... page fault

READ e8001 ... page fault

READ ec001 ... page fault

READ f0001 ... page fault

READ f4001 ... page fault

READ f8001 ... page fault

READ fc000 ... page fault