

# **Operating Systems (EOPSY)**

Laboratory no. 4

*Memory management*

Laura Ploch

July 2021

## 1. Introduction

**Memory management** in operating systems is the function responsible for managing the computer's memory. It keeps track of the status of each memory location, whether it's allocated or free. It decides which process gets memory, when and how much.

**Virtual memory** is a memory management technique which combines the storage resources into an idealized abstraction of memory, where virtual memory addresses are mapped into physical addresses in computer memory. This allows the system to conceptually use more memory than might be physically available, using the technique of segmentation or paging.

In our case – we observe the **paging** scheme of memory management. This method divides the computer's memory into fixed-size called page frames and the virtual address space of the program into same-size blocks called pages.

**Page fault** occurs when a process tries to reference a page not currently present in RAM. Such invalid memory reference is treated by the processor as a page fault and then the control is transferred from the program to the operating system, which then:

- Determines the location of the data on disk
- Gets an empty page in RAM to use as a container for the data
- Loads the data into the available page frame (or a retrieved one, selected to reuse)
- Updates the page table to refer to the new page frame
- Returns control to the program

## 2. Task

The aim of this laboratory task was to configure Memory Management simulator so that it maps 8 pages of physical memory to the first 8 pages of virtual memory and then on each of the 64 virtual pages read from one virtual memory address. To do this the *memory.conf* and *commands* files were to be modified.

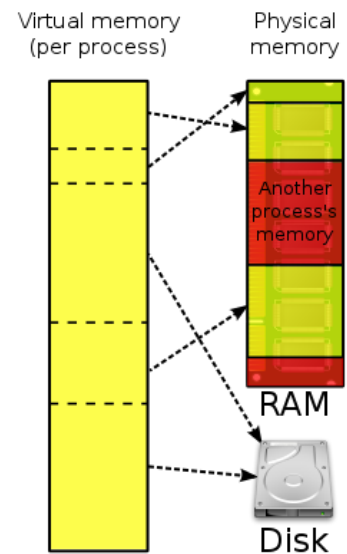


Fig. 1 Virtual memory mapping

```

1 // memset virt page # physical page # R (read from) M (modified) inMemTime (ns)
  lastTouchTime (ns)
2 memset 0 0 0 0 0 0
3 memset 1 1 0 0 0 0
4 memset 2 2 0 0 0 0
5 memset 3 3 0 0 0 0
6 memset 4 4 0 0 0 0
7 memset 5 5 0 0 0 0
8 memset 6 6 0 0 0 0
9 memset 7 7 0 0 0 0
10
11
12 // enable_logging 'true' or 'false'
13 // When true specify a log_file or leave blank for stdout
14 enable_logging true
15
16 // log_file <FILENAME>
17 // Where <FILENAME> is the name of the file you want output
18 // to be print to.
19 log_file tracefile
20
21 // page size, defaults to 2^14 and cannot be greater than 2^26
22 // pagesize <single page size (base 10)> or <'power' num (base 2)>
23 pagesize 16384
24
25 // addressradix sets the radix in which numerical values are displayed
26 // 2 is the default value
27 // addressradix <radix>
28 addressradix 10
29
30 // numpages sets the number of pages (physical and virtual)
31 // 64 is the default value
32 // numpages must be at least 2 and no more than 64
33 // numpages <num>
34 numpages 64

```

Fig. 2.1 *memory.conf* file

```

1 // Enter READ/WRITE commands into this file
2 // READ <OPTIONAL number type: bin/hex/oct> <virtual memory address or random>
3 // WRITE <OPTIONAL number type: bin/hex/oct> <virtual memory address or random>
4 READ 0 // address to read
5 READ 16384 // the multiples of 16384
6 READ 32768
7 READ 49152
8 READ 65536
9 READ 81920
  (...)
58 READ 884736
59 READ 901120
60 READ 917504
61 READ 933888
62 READ 950272
63 READ 966656
64 READ 983040
65 READ 999424
66 READ 1015808
67 READ 1032192

```

Fig. 2.2 *commands* file

The *memory.conf* was edited so that the 8 pages of physical memory are mapped to the first 8 pages of virtual memory using *memset* command. The physical pages are mapped consecutively, so that the results of the simulation are clear and easier to be observed.

Additionally I changed the *addressradix* to 10 to set the radix in which the numerical values are displayed to decimal.

In *commands* file the command *READ* + address was repeated 64 times with addresses that are multiples of the size of the page – 16384, so that each time we read virtual memory address from each of the 64 virtual pages.

### 3. Simulation results

virtual	physical	virtual	physical	time: 0
page 0	page 0	page 32		
page 1	page 1	page 33		instruction: NONE
page 2	page 2	page 34		address: NULL
page 3	page 3	page 35		
page 4	page 4	page 36		page fault: NO
page 5	page 5	page 37		
page 6	page 6	page 38		virtual page: x
page 7	page 7	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		

Fig. 3.1 Mapping before the start of the simulation

virtual	physical	virtual	physical	time: 500 (ns)	virtual	physical	virtual	physical	time: 640 (ns)
page 0		page 32	page 0		page 0		page 32	page 0	
page 1		page 33	page 1	instruction: READ	page 1		page 33	page 1	instruction: READ
page 2		page 34	page 2	address: 802816	page 2		page 34	page 2	address: 1032192
page 3		page 35	page 3		page 3		page 35	page 3	
page 4		page 36	page 4	page fault: YES	page 4		page 36	page 4	page fault: YES
page 5		page 37	page 5		page 5		page 37	page 5	
page 6		page 38	page 6	virtual page: 49	page 6		page 38	page 6	virtual page: 63
page 7		page 39	page 7	physical page: -1	page 7		page 39	page 7	physical page: 31
page 8		page 40	page 8	R: 0	page 8		page 40	page 8	R: 0
page 9		page 41	page 9	M: 0	page 9		page 41	page 9	M: 0
page 10		page 42	page 10	inMemTime: 0	page 10		page 42	page 10	inMemTime: 0
page 11		page 43	page 11	lastTouchTime: 0	page 11		page 43	page 11	lastTouchTime: 0
page 12		page 44	page 12	low: 802816	page 12		page 44	page 12	low: 1032192
page 13		page 45	page 13	high: 819199	page 13		page 45	page 13	high: 1048575
page 14		page 46	page 14		page 14		page 46	page 14	
page 15		page 47	page 15		page 15		page 47	page 15	
page 16		page 48	page 16		page 16		page 48	page 16	
page 17		page 49	page 17		page 17		page 49	page 17	
page 18	page 18	page 50			page 18		page 50	page 18	
page 19	page 19	page 51			page 19		page 51	page 19	
page 20	page 20	page 52			page 20		page 52	page 20	
page 21	page 21	page 53			page 21		page 53	page 21	
page 22	page 22	page 54			page 22		page 54	page 22	
page 23	page 23	page 55			page 23		page 55	page 23	
page 24	page 24	page 56			page 24		page 56	page 24	
page 25	page 25	page 57			page 25		page 57	page 25	
page 26	page 26	page 58			page 26		page 58	page 26	
page 27	page 27	page 59			page 27		page 59	page 27	
page 28	page 28	page 60			page 28		page 60	page 28	
page 29	page 29	page 61			page 29		page 61	page 29	
page 30	page 30	page 62			page 30		page 62	page 30	
page 31	page 31	page 63			page 31		page 63	page 31	

Fig. 3.2 Mapping during the simulation

Fig. 3.3 Final mapping at the end of the simulation

1 READ 0 ... okay	33 READ 524288 ... page fault
2 READ 16384 ... okay	34 READ 540672 ... page fault
3 READ 32768 ... okay	35 READ 557056 ... page fault
4 READ 49152 ... okay	36 READ 573440 ... page fault
5 READ 65536 ... okay	37 READ 589824 ... page fault
6 READ 81920 ... okay	38 READ 606208 ... page fault
7 READ 98304 ... okay	39 READ 622592 ... page fault
8 READ 114688 ... okay	40 READ 638976 ... page fault
9 READ 131072 ... okay	41 READ 655360 ... page fault
10 READ 147456 ... okay	42 READ 671744 ... page fault
11 READ 163840 ... okay	43 READ 688128 ... page fault
12 READ 180224 ... okay	44 READ 704512 ... page fault
13 READ 196608 ... okay	45 READ 720896 ... page fault
14 READ 212992 ... okay	46 READ 737280 ... page fault
15 READ 229376 ... okay	47 READ 753664 ... page fault
16 READ 245760 ... okay	48 READ 770048 ... page fault
17 READ 262144 ... okay	49 READ 786432 ... page fault
18 READ 278528 ... okay	50 READ 802816 ... page fault
19 READ 294912 ... okay	51 READ 819200 ... page fault
20 READ 311296 ... okay	52 READ 835584 ... page fault
21 READ 327680 ... okay	53 READ 851968 ... page fault
22 READ 344064 ... okay	54 READ 868352 ... page fault
23 READ 360448 ... okay	55 READ 884736 ... page fault
24 READ 376832 ... okay	56 READ 901120 ... page fault
25 READ 393216 ... okay	57 READ 917504 ... page fault
26 READ 409600 ... okay	58 READ 933888 ... page fault
27 READ 425984 ... okay	59 READ 950272 ... page fault
28 READ 442368 ... okay	60 READ 966656 ... page fault
29 READ 458752 ... okay	61 READ 983040 ... page fault
30 READ 475136 ... okay	62 READ 999424 ... page fault
31 READ 491520 ... okay	63 READ 1015808 ... page fault
32 READ 507904 ... okay	64 READ 1032192 ... page fault

Fig. 3.4 output of the simulation in the *tracefile*

```

1 /* It is in this file, specifically the replacePage function that will
2    be called by MemoryManagement when there is a page fault. The
3    users of this program should rewrite PageFault to implement the
4    page replacement algorithm.
5 */
6
7 // This PageFault file is an example of the FIFO Page Replacement
8 // Algorithm as described in the Memory Management section.
9
10 import java.util.*;
11
12 public class PageFault {
13
14     /**
15      * The page replacement algorithm for the memory management simulator.
16      * This method gets called whenever a page needs to be replaced.
17      * <p>
18      * The page replacement algorithm included with the simulator is
19      * FIFO (first-in first-out). A while or for loop should be used
20      * to search through the current memory contents for a candidate
21      * replacement page. In the case of FIFO the while loop is used
22      * to find the proper page while making sure that virtPageNum is
23      * not exceeded.
24      * <pre>
25      * Page page = ( Page ) mem.elementAt( oldestPage )
26      * </pre>
27      * This line brings the contents of the Page at oldestPage (a
28      * specified integer) from the mem vector into the page object.
29      * Next recall the contents of the target page, replacePageNum.
30      * Set the physical memory address of the page to be added equal
31      * to the page to be removed.

```

Fig. 3.5 *PageFault.java* file

## 4. Observations

Stepping through operations during the simulation we could observe that first 8 pages that we specified in *memory.conf* file were mapped and read correctly. Moreover, the pages up to 31 (32 out of 64) also didn't produce any errors, since the mapping was probably done by default. This can be also observed in the output *tracefile* file (fig. 3.4)

However, on the 33<sup>rd</sup> page we started encountering page faults, since these page has not been mapped to any physical pages. As described in the introduction, the problem was solved by mapping consecutive virtual pages to the previously mapped physical pages (which can be observed on figures 3.1-3.3)

The algorithm of page replacement was observed to be FIFO – first in, first out – the first mapped physical page (since no free pages were available) was reused and mapped to the first virtual page on which page fault occurred. This can also be confirmed in *PageFault.java* file (fig. 3.5) where it is clearly stated that the page replacement algorithm used by the simulator is FIFO (first-in first-out).