EOPSY Laboratory 4

Jakub Tomkiewicz 300183

In this laboratory we are working with MOSS Memory Management Simulator, which explains page fault behaviour in a paged virtual memory system. Simulator reads the initial state of the page table and a sequence of virtual memory instructions and writes trace log that indicates the effect of each instruction.

Analysing java files that came with the simulator shows that the included page replacement algorithm is FIFO (first-in first-out). We are able to see it in comments of PageFault.java.

```
12  public class PageFault {
13
14    /**
15     * The page replacement algorithm for the memory management sumulator.
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 canidate
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.
```

FIFO is the simplest page replacement algorithm. It first executes the job that entered the queue first. It was mentioned in the laboratory 3, when speaking about First-Come First-Served scheduling name.

Our task in laboratory number 4 is to write command file that maps 8 pages of physical memory to the first 8 pages of the virtual memory, and then reads from one virtual memory address on each of the 64 virtual pages.

In order to properly perform laboratory we need to edit two files. First one is `memory.conf`, which is used to specify the content of the virtual memory map. It tells which pages of virtual memory are mapped to which pages of the physical memory.

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

We are working on (only) 8 pages, so we need to map only 8 pages with memset instruction, which is used to initialise entries of the virtual map page. In order to make the laboratory more interesting, I changed the second value in memset instruction, so the physical page associated with virtual page. As default virtual page 0 maps to physical page 0, virtual 1 maps to physical 1 and so on.

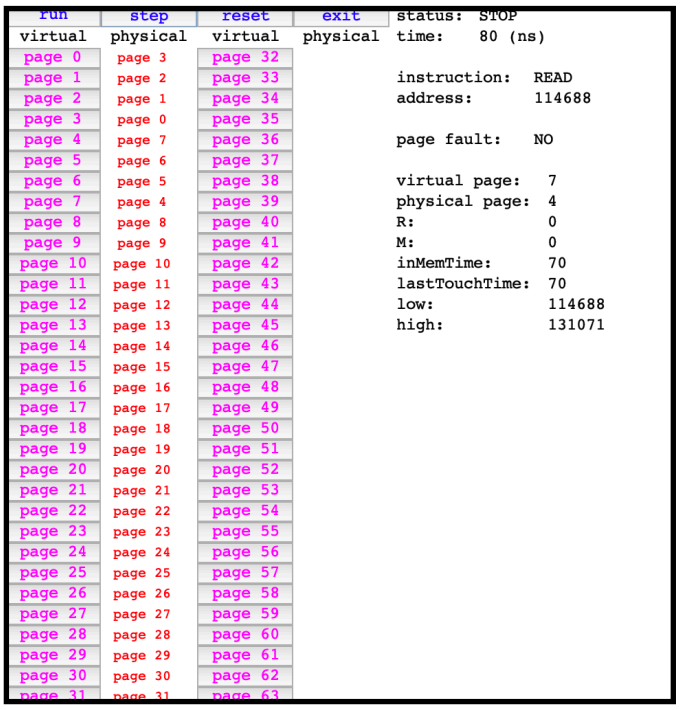
To make my work with numbers a little bit easier, I changed addressradix number to 10 in order to work on decimal numbers. As default it is set to 16, but for example in can be set to 2 if someone would like to work on binary numbers.

Second file that needs to be changed before compiling code is `commands`. It specifies a sequence of memory instructions to be performed. READ and WRITE memory operations can be performed on virtual memory addresses.

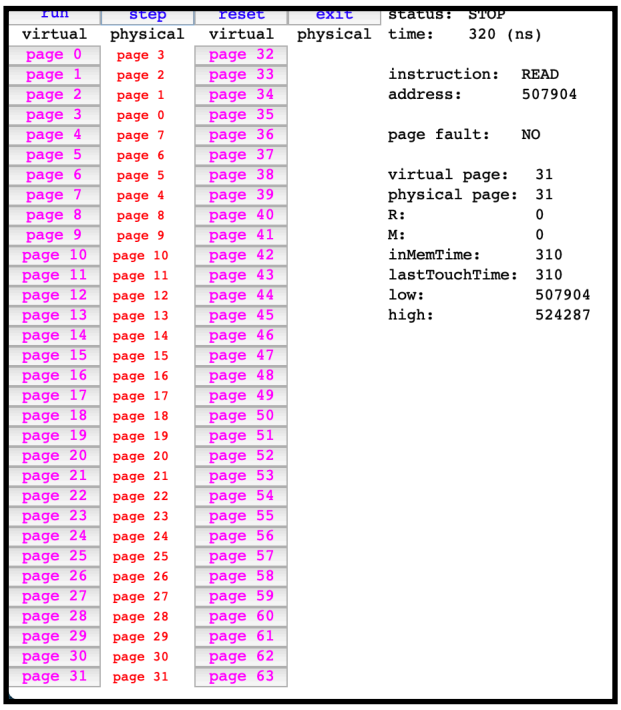
```
1   // Enter READ/WRITE commands into this file
2   // READ <OPTIONAL number type: bin/hex/oct> <virtual memory address or random>
3   // WRITE <OPTIONAL number type: bin/hex/oct> <virtual memory address or random>
4   //64 addresses increasing by 16384
5   READ 0
6   READ 16384
7   READ 32768
8   READ 49152
9   READ 65536
10  READ 81920
11  READ 98304
12  ...
13  ...
14  ...
15  READ 999424
16  READ 1015808
17  READ 1032192
```

According to our instruction we want to read one virtual memory address 64 times. To achieve it command READ needs to be performed 64 times on addresses differing by 16384 in order to read one address of virtual memory. 16384 is chosen as it is the default pagesize in `memory.conf` file. It can changed, but to maximal value of 67108864.

According to laboratory instruction I stepped through the simulation step after step. First 8 steps mapped 8 pages correctly as it was specified in `memory.conf` file. Moreover, pages from 0 to 31 were all mapped correctly.

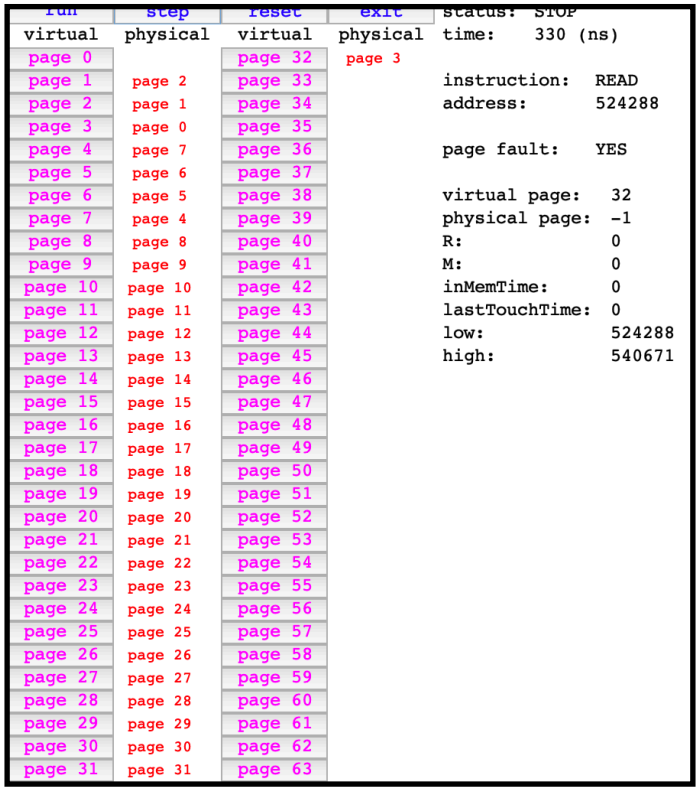


Page 7



Page 31

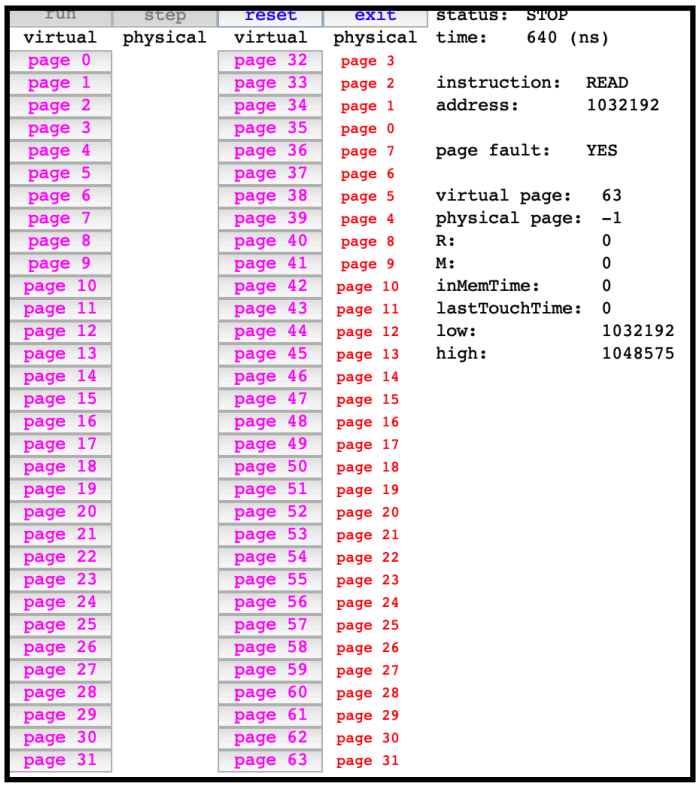
On page 32 occurs page fault. Program tries to map virtual page 32 to not specified physical page and shows physical page number as -1.



run	step	reset	exit	status: STOP
virtual	physical	virtual	physical	time: 330 (ns)
page 0		page 32	page 3	
page 1	page 2	page 33		instruction: READ
page 2	page 1	page 34		address: 524288
page 3	page 0	page 35		
page 4	page 7	page 36		page fault: YES
page 5	page 6	page 37		
page 6	page 5	page 38		virtual page: 32
page 7	page 4	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: 524288
page 13	page 13	page 45		high: 540671
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		

FIFO algorithm on page fault takes the page that is the longest in memory, in our example page 3, and replaces it. Virtual page 32 has taken physical page 3 as a replacement.

Pages from 32 to 63 shows page fault with physical pages number set to -1 as in page 32.



run	step	reset	exit	status: STOP
virtual	physical	virtual	physical	time: 640 (ns)
page 0		page 32	page 3	
page 1		page 33	page 2	instruction: READ
page 2		page 34	page 1	address: 1032192
page 3		page 35	page 0	
page 4		page 36	page 7	page fault: YES
page 5		page 37	page 6	
page 6		page 38	page 5	virtual page: 63
page 7		page 39	page 4	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: 1032192
page 13		page 45	page 13	high: 1048575
page 14		page 46	page 14	
page 15		page 47	page 15	
page 16		page 48	page 16	
page 17		page 49	page 17	
page 18		page 50	page 18	
page 19		page 51	page 19	
page 20		page 52	page 20	
page 21		page 53	page 21	
page 22		page 54	page 22	
page 23		page 55	page 23	
page 24		page 56	page 24	
page 25		page 57	page 25	
page 26		page 58	page 26	
page 27		page 59	page 27	
page 28		page 60	page 28	
page 29		page 61	page 29	
page 30		page 62	page 30	
page 31		page 63	page 31	

FIFO algorithm assigned for these pages oldest possible, at this exact moment, page in memory. For virtual page 33 physical page 2 was found as a replacement, for virtual 34 physical 3 and so on until virtual page 63 gets physical page 31 as a replacement.

`Tracefile` shows exact moment of the first page fault at address 524288, as mentioned before, on page 32.

```
READ 0 ... okay
READ 16384 ... okay
READ 32768 ... okay
READ 49152 ... okay
READ 65536 ... okay
READ 81920 ... okay
...
...
...
READ 458752 ... okay
READ 475136 ... okay
READ 491520 ... okay
READ 507904 ... okay
READ 524288 ... page fault
READ 540672 ... page fault
READ 557056 ... page fault
READ 573440 ... page fault
...
...
...
READ 999424 ... page fault
READ 1015808 ... page fault
READ 1032192 ... page fault
```