

EOPSY Lab Task 4 – Filip Wasilewski

Setting

We have a simulator of memory management simulator. We can configure it using two files: commands, and memory.conf. The results of our simulation are displayed in the graphical UI provided, as well as the result log file „traceline”.

run	step	reset	exit	status: STOP	
virtual	physical	virtual	physical	time: 0	
page 0	page 0	page 32		instruction: NONE	
page 1	page 1	page 33		address: NULL	
page 2	page 2	page 34		page fault: NO	
page 3	page 3	page 35		virtual page: x	
page 4	page 4	page 36		physical page: 0	
page 5	page 5	page 37		R: 0	
page 6	page 6	page 38		M: 0	
page 7	page 7	page 39		inMemTime: 0	
page 8	page 8	page 40		lastTouchTime: 0	
page 9	page 9	page 41		low: 0	
page 10	page 10	page 42		high: 0	
page 11	page 11	page 43			
page 12	page 12	page 44			
page 13	page 13	page 45			
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			

UI example

Clicking on virtual page number gives us in the parameters some information, the same as will be described in input files. For now i want to draw our attention to parameter physical page. It will give us number of page mapped to the selected virtual one.

The input files

This is how example file commands looks like:

READ bin 100

```
READ 19
WRITE hex CC32
READ bin 1000000000000000
READ bin 1000000000000000
WRITE bin 1100000000000001
WRITE random
```

The commands pattern, which are READ/WRITE commands from specific address have a pattern:

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

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

The file memory.conf is a bit more complex. It looks like (without comments):

```
memset 0 0 0 0 0
memset 1 1 0 0 0
(there is 32 of these memset lines)

enable_logging true
log_file tracefile
pagesize 16384
addressradix 16
numpages 64
```

memset sets the virtual and physical page, the numbers next to it are:

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

enable_logging – ['true'/'false'] when true specify a log_file or leave blank for stdout. You specify name below in logfile property for the results to be printed there.

page size - defaults 2^{14} , can't be greater than 2^{26}

addressradix - sets the base in which numerical values are displayed

numpages – sets the number of pages (physical and virtual), defaults to 64, has to be at least 2 (because then we have one physical one virtual), and not more than 64

Result file

„Tracefile” has logs of operations, which were conducted. There is not a lot to look at here. The example logs look like this:

```
-----  
READ 4 ... okay  
READ 13 ... okay  
WRITE cc32 ... okay  
READ 4000 ... okay  
READ 4000 ... okay  
WRITE 6001 ... okay  
WRITE 78a4e ... okay  
-----
```

So we get update how the specific operation performed.

Task objective

The task is as follows:

1. Create a command file that maps any 8 pages of physical memory to the first 8 pages of virtual memory
2. Then the program reads from one virtual memory address on each of the 64 virtual pages.
3. We should step through the simulator one operation at a time and see if you can predict which virtual memory addresses cause page faults.
4. Last task is to determine page replacement algorithm is being used?

1, 2, 3 - Memory mapping and reading

Those tasks combined proved to be impossible to conduct! Why? Because numpages sets the number of pages always to be equal to each other. So when we want to map 8 physical pages, we need to have 8 virtual ones.

But the task is to read from all 64 of virtual pages, which is not possible even for 64 physical pages. Because numpages is a sum of those two types, we can have at most 32 pages virtually mapped to 32

pages physically (we can determine that because for max value of numpages, 64, only 32 physical pages are created and mapped to 32 virtual pages).

It does not mean we can't read from virtual page 32. It is that we immediately know when the program will return page fault – when we reach the page incorrectly mapped, when page 32 (i remind, because numeration starts from 0) will be operated on, due to the fact, that in it's physical page mapping parameter, in the program we have -1.

I will show in the next point, that the first page to fault will be the first one not mapped properly, so page 33, 32 in program

4 – Page replacement algorithm

As an example, i will have two instructions in commands, one reads from the address, which would be page number 33 (numeration starts from 0), having mapped max of 32 pages. The other one will read from page 0. It will allow us to determine how pages are replaced.

This is how our commands file looks like:

```
-----  
WRITE hex 80000  
WRITE hex 1  
-----
```

Where 0x80000 is the address in virtual page 32

Let's look once more at the starting situation:

run	step	reset	exit	STATUS: SIUP
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		

Then, when we execute the first instruction, surprise! Because there could be mapped max of 32 pages, 33rd (denoted here as 32) was not mapped, when we wrote to the address beyond mapped ones, the page faulted and we needed to use the algorithm to replace the page. Virtual page 33 is now mapped to physical page 0, whereas virtual 0 has no mapping.

There appears log in tracefile:

WRITE 80000 ... page fault

run	step	reset	exit	status: STOP
virtual	physical	virtual	physical	time: 10 (ns)
page 0		page 32	page 0	instruction: WRITE
page 1	page 1	page 33		address: 80000
page 2	page 2	page 34		page fault: YES
page 3	page 3	page 35		virtual page: 32
page 4	page 4	page 36		physical page: -1
page 5	page 5	page 37		R: 0
page 6	page 6	page 38		M: 0
page 7	page 7	page 39		inMemTime: 0
page 8	page 8	page 40		lastTouchTime: 0
page 9	page 9	page 41		low: 80000
page 10	page 10	page 42		high: 83fff
page 11	page 11	page 43		
page 12	page 12	page 44		
page 13	page 13	page 45		
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		

Going further, we see that with writing to page 0 now gives another page fault, another replacement happens:

run	stop	reset	exit	status: stop	
virtual	physical	virtual	physical	time: 20 (ns)	
page 0	page 1	page 32	page 0	instruction: WRITE	
page 1		page 33		address: 1	
page 2	page 2	page 34			
page 3	page 3	page 35		page fault: YES	
page 4	page 4	page 36		virtual page: 0	
page 5	page 5	page 37		physical page: 1	
page 6	page 6	page 38		R: 0	
page 7	page 7	page 39		M: 0	
page 8	page 8	page 40		inMemTime: 10	
page 9	page 9	page 41		lastTouchTime: 10	
page 10	page 10	page 42		low: 0	
page 11	page 11	page 43		high: 3fff	
page 12	page 12	page 44			
page 13	page 13	page 45			
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			

As the file PageFault.java confirms, the algorithm used to replace pages is FIFO – First In First Out.

It keeps pages in a queue, from oldest to newest. When a replacement is called, front of queue is the one chosen to be removed, to assign somewhere where needed.

This explains, why the first page to be remapped was page 0, and consequently page 1 became next.