EOPSY

Lab 4

Michał Kurczak

Task Description

Create a command file that maps any 8 pages of physical memory to the first 8 pages of virtual memory, and then reads from one virtual memory address on each of the 64 virtual pages. Step through the simulator one operation at a time and see if you can predict which virtual memory addresses cause page faults. What page replacement algorithm is being used?

Background

Computer memories are made up of registers. Each register comprises of cells capable of storing one bit data, address lines used for routing the data by means of data buses. The matter of whether we want to write to the memory or read from it is handled by read and write connection lines. Those register make a one dimensional array which is used for storing data and processes. The simplest way to write them into the memory would be to organize them in a contiguous manner so right after the last bit of one program we write the first bit of another one. But processes come and go, hence when we free a memory a vacant space is left. Fortunately, a device called Memory Management Unit keeps track of it and allocates the free memory space. If we want to access the memory that has no physical address mapped then a page fault warning is returned.

But processes differ in sizes. This problem is partially handled by partitioning the memory into segments called pages.

In terms of our exercise, we are dealing with the static partitioning, where the size of each partition is pre-set. The "pagesize" variable in the memory config of our program has been set to the default 16384 and number of pages has been set to 64

```
// 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
// 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
```

Figure 1. Initial configurations

The task is to map the first 8 virtual addresses to the physical addresses but the application is not working properly so we'll map all the 32 pages with default settings:

Figure 2. Initial configuration of memory mapping

Using simple bash script

Figure 3. Automatic commands generation

a command file is made allowing us to read all the 64 pages status at each step

```
1 READ 0
2 READ 16384
3 READ 32768
4 READ 49152
5 READ 65536
6 READ 81920
7 READ 98304
8 READ 114688
9 READ 131072
```

Figure 4. Snipped of the commands file

After each **READ** call I'm specifying address of **n'th** page given by the formula:

```
page_address = n * page_size
```

As the reading of the corresponding pages proceeds we may predict that the first page fault will occur at the virtual page 32 as it is initially the first page with unmapped physical address.

	alemai a a 1		abuai aal	44	^	
virtual	physical	virtual	physical	time:	0	
page 0	_	page 32		24	.	NONE
page 1	page 0	page 33		instruc		NONE
page 2	page 1	page 34		address	:	NULL
page 3	page 2	page 35		,	٠.	
page 4	page 3	page 36		page fa	ult:	NO
page 5	page 4	page 37				
page 6	page 5	page 38 I		virtual		X
page 7	page 6	page 39		physica	l page	
page 8	page 8	page 40		R:		0
page 9	page 9	page 41		М:		0
_page_10_	page 10	page 42		inMemTi		0
_page_ll_	page 11	page 43		lastTou	chTime	
_page_12_	page 12	page 44		low:		0
page 13	page 13	page 45 l		high:		0
page 14	page 14	page 46				
page 15	page 15	_page_47_				
page 16	page 16	page 48 l				
page 17	page 17	page 49				
page 18	page 18	page 50 l				
page 19	page 19	page 51				
page 20 I	page 20	page 52				
page 21	page 21	page 53 l				
page 22 I	page 22	page 54				
page 23 I	page 23	page 55 l				
page 24	page 24	page 56 l				
page 25 I	page 25	page 57 I				
page 26	page 26	page 58 I				
page 27	page 27	page 59 l				
page 28	page 28	page 60 l				
page 29	page 29	page 61				
page 30	page 30	page 62 l				
nane 31 (page 31	пале 63 (

Figure 5. Initial state of the program

virtual	physical	virtual	physical	time: 330 (ns)
page 0		page 32	page 1	
_page 1	page 0	page 33		instruction: READ
page 2	page 1	page 34		address: 80000
page 3	page 2	page 35		6 11 250
_page_4_	page 3	page 36		page fault: YES
page 5	page 4	page 37		
page 6	page 5	page 38		virtual page: 32
page 7	page 6	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_ll_	page 11	page 43		lastTouchTime: 0
page 12	page 12	page 44		low: 80000
page 13	page 13	page 45		high: 83fff
_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 l	page 30	page 62		
page 31	page 31	page 63 l		

Figure 6. First page fault occurence

The page fault indeed occurred. Let's have a look at the "tracefile" output obtained from the configuration file

```
27 READ 68000 ... okay
28 READ 6C000 ... okay
29 READ 70000 ... okay
30 READ 74000 ... okay
31 READ 78000 ... okay
32 READ 7C000 ... okay
33 READ 80000 ... page fault
34 READ 84000 ... page fault
35 READ 88000 ... page fault
36 READ 8C000 ... page fault
37 READ 90000 ... page fault
```

Figure 7. Tracefile output

The first page fault occurred at line 33 which corresponds to the address of the virtual page with index 32. As the program executes we observe that right after we try to read the virtual page 32 it

gets the first physical address mapped. We can conclude that most likely the algorithm for memory management is a simple FIFO queue. Let's try to read from virtual page 32 right after new memory space is allocated by pasting the same line twice.

Address of the virtual page 32

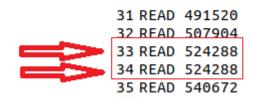


Figure 8. Reading the page after memory mapping

First page fault occurs

instruction: READ address: 80000 page fault: YES virtual page: physical page: -1 R: 0 M: 0 inMemTime: 0 lastTouchTime: 0 low: 80000 high: 83fff

Figure 9. First execution of READ

instruction: READ

Then when we go to the next step

address: 80000 page fault: NO virtual page: physical page: 1 R: 0 M: 0 inMemTime: 10 lastTouchTime: 10 low: 80000 high: 83fff

Figure 10. Second execution of READ

No page fault occurred since memory has been correctly mapped.

Tuning the program up

Now let's try a different number of pages and try to change the default mapping config by addressing to the first 8 virtual addresses corresponding physical addresses but incremented by 1 - no physical page 0 is present

```
1 // memset virt page # physical page # R (read from) M (modified) inMemTime (ns) lastTouchTime (ns)
2 memset 0 1 0 0 0 0 0
3 memset 1 2 0 0 0 0 0
4 memset 2 3 0 0 0 0 0
5 memset 3 4 0 0 0 0 0
6 memset 4 5 0 0 0 0 0
8 memset 5 6 0 0 0 0 0
9 memset 7 8 0 0 0 0

// 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 32
```

Figure 11. Initial configurations

The first thing we can notice that the physical page 0 is still available even though we specified it in another way. On the other hand the physical page 7 got lost, but when we inspect a page the config seems ok

virtual	physical	virtual	physical	time: 0	
page 0		page 32			
page 1	page 0	page 33 I		instruction	: NONE
page 2	page 1	page 34 I		address:	NULL
page 3	page 2	page 35 I			
page 4	page 3	page 36 l		page fault:	NO
page 5	page 4	_page_37_1			
page 6	page 5	page 38 I		virtual pag	
page 7	page 6	page 39 l		physical pa	
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_ll_	page 11	page 43		lastTouchTi	
page 12	page 12	page 44		low:	0
page 13	page 13	page 45		high:	3fff
page 14	page 14	page 46			
page 15	page 15	page 47			
page 16		page 48			
page 17		page 49			
page 18		page 50			
page 19		page 51			
page 20		page 52			
page 21		page 53			
page 22		page 54			
page 23		page 55			
page 24		page 56			
page 25		page 57			
page 26		page 58			
page 27 I		page 59 I			
page 28 I		page 60			
page 29 I		page 61			
page 30 l		page 62			
page 31		page 63			

Figure 12. Initial state of the program

As we could predict, the first page fault occurred on the page 16 but now we see that two physical page 1 occurred

virtual	physical	virtual	physical	time: 170 (r	ns)
page 0		page 32 l			
page 1	page 0	page 33 I		instruction:	READ
page 2	page 1	page 34 I		address:	40000
page 3	page 2	page 35 I			
page 4	page 3	page 36 l		page fault:	YES
page 5	page 4	page 37 I			
page 6	page 5	page 38 I		virtual page:	16
page 7	page 6	page 39 I		physical page:	1
page 8	page 8	page 40 l		R:	0
page 9	page 9	_page_41_		M:	0
page 10	page 10	page 42 l		inMemTime:	10
page 11	page 11	page 43 l		lastTouchTime:	10
page 12	page 12	page 44		low:	40000
page 13	page 13	page 45		high:	43fff
page 14	page 14	page 46			
page 15	page 15	page 47			
page 16	page 1	page 48			

Figure 13. First page fault occurence

But when we investigate it furtherly it is just a mater of graphical user interface hardcoded values than the simulation backend code

virtual	physical	virtual	physical	time: 170 (ns)
page 0		page 32		
_page 1	page 0	page 33 l		instruction: READ
page 2	page 1	page 34		address: 40000
page 3	page 2	page 35 l		
page 4	page 3	page 36 l		page fault: YES
page 5	page 4	page 37 l		
page 6	page 5	page 38 I		virtual page: 1
page 7	page 6	page 39 l		physical page: 2
page 8	page 8	page 40		R: 0
page 9	page 9	page 41		M: 0
page 10	page 10	page 42		inMemTime: 170
page 11	page 11	page 43 l		lastTouchTime: 160
page 12	page 12	page 44		low: 4000
page 13	page 13	page 45		high: 7fff
page 14	page 14	page 46		
page 15	page 15	page 47		
page 16	page 1	page 48 I		

Figure 14. Mapping check

Yet, looking at the virtual page 7 and 8 we can notice that both of them share the same physical memory

instruction: READ instruction: READ address: 40000 address: 40000 page fault: YES page fault: YES virtual page: virtual page: physical page: 8 physical page: 8 R: 0 R: M: 0 M: inMemTime: 170 inMemTime: 170 lastTouchTime: 100 lastTouchTime: 90 low: 1c000 low: 20000 1ffff high: high: 23fff

Conclusions

Using the memory management simulation, one can possess a better understanding on how this hardware works, yet has to be careful because a lots of improvement should be done in the matters of configuration.

Resources

Geeks for geeks: https://www.geeksforgeeks.org/paging-in-operating-system/

Tomasz Kruk's EOPSY website: http://elektron.elka.pw.edu.pl/~tkruk/edu/eopsy/