

EOPSY LAB 4

Krzysztof Piotrowski

Introduction

Our Task is to map any 8 pages of physical memory to the first 8 pages of virtual memory and reads from one virtual memory address on each of the virtual pages. For this we will use Memory Management simulator. Simulator consists of 64 virtual pages and 32 physical pages. Each page is defined as 16384B by default.

Configuration

Two files need to be configured: memory.conf and commands. Here are its content defined for our task:

memset 0 0 0 0 0	READ hex 00000
memset 1 5 0 0 0	READ hex 04000
memset 2 10 0 0 0	READ hex 08000
memset 3 15 0 0 0	READ hex 0C000
memset 4 20 0 0 0	READ hex 10000
memset 5 25 0 0 0	READ hex 14000
memset 6 30 0 0 0	.
memset 7 31 0 0 0	.
	.
enable_logging true	READ hex 40000
log_file tracefile	READ hex 44000
pagesize 16384	READ hex 48000
addressradix 16	READ hex 4C000
numpages 64	READ hex 50000
	READ hex 54000
	READ hex 58000
	READ hex 5C000
	READ hex F4000
	READ hex F8000
	READ hex FC000

As it can be seen above I randomly assign physical memory pages to virtual memory pages with the use of command memset, where first column stand for virtual page and second for physical page. Rest of columns does not matter in our exercise.

On the right size I defined commands for reading each of 64 virtual memory pages. Values go from 0 to FC000 (hex) and change by 4000 (hex) as it is size of our default page 16384 B in hexadecimal form.

Simulation

When program was ran, the following view was seen:



At first glance It looks like there is some trouble with our page assignment. However, when we look deeply into virtual description we can see assigned physical page (In that case 1st virtual page has assigned 5th physical page).

Before starting simulation I can already predict that „page fault” will occur after 31st page. We have 64 virtual pages and 32 physical pages, When program will try to reference virtual page without physical page mapping, the page replacement algorithm will start its work.

```

READ 0 ... okay
READ 4000 ... okay
READ 8000 ... okay
READ c000 ... okay
.
.
READ 70000 ... okay
READ 74000 ... okay
READ 78000 ... okay
READ 7c000 ... okay
READ 80000 ... page fault
READ 84000 ... page fault
READ 88000 ... page fault
READ 8c000 ... page fault
.
.
READ f0000 ... page fault
READ f4000 ... page fault
READ f8000 ... page fault
READ fc000 ... page fault

```

The screenshot shows a 'Memory Management' window with a list of virtual and physical pages on the left and a detailed view of a page fault on the right.

virtual	physical	virtual	physical
page 0		page 32	page 0
page 1		page 33	page 5
page 2		page 34	page 10
page 3		page 35	page 15
page 4		page 36	page 20
page 5		page 37	page 25
page 6		page 38	page 30
page 7		page 39	page 31
page 8		page 40	page 8
page 9		page 41	page 9
page 10		page 42	page 10
page 11		page 43	page 11
page 12		page 44	page 12
page 13		page 45	page 13
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

status: STOP
time: 640 (ns)

instruction: READ
address: fc000

page fault: YES

virtual page: 63
physical page: -1
R: 0
M: 0
inMemTime: 0
lastTouchTime: 0
low: fc000
high: fffff

As it can be seen above we see from which page we got page fault (from tracefile) and the way algorithm assign physical pages (program screen). According to my prediction page fault begins from 80000 (hex) address which stands for 32 page and goes up to the last virtual page. Starting from page 32 and going in order, algorithm assign following physical pages: 0, 5, 10, 15 etc. These are pages that we assign to first 8 virtual pages. Thanks to that information we know if there is need for page mapping, algorithm assign physical pages from the oldest virtual page. It means that page replacement algorithm base on **First in First Out** algorithm. System is tracking every page and when there is need for page replacement it takes first one from the queue which in our case is the oldest one.