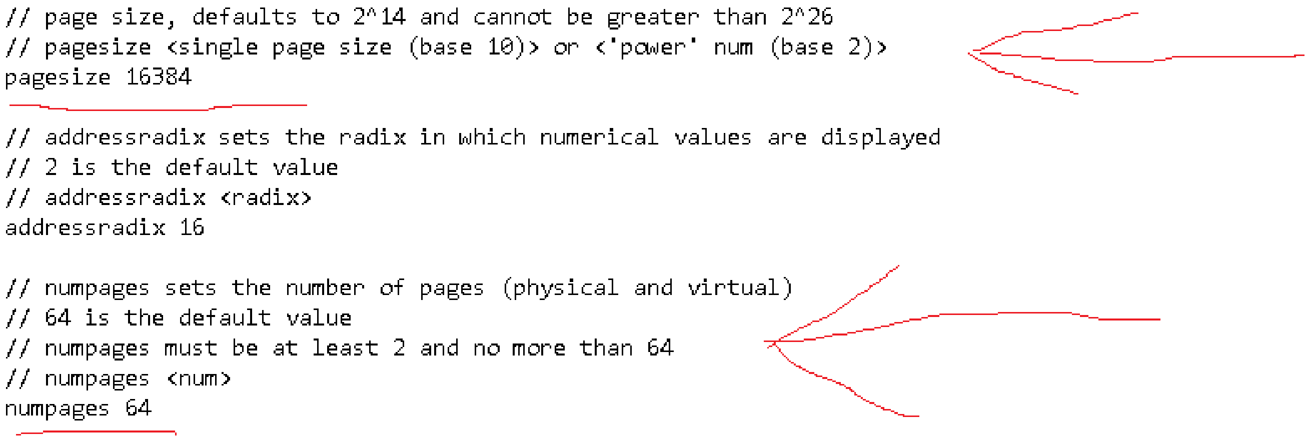
EOPSY – Task 4 **Abbaszade Anvar**

Task description.

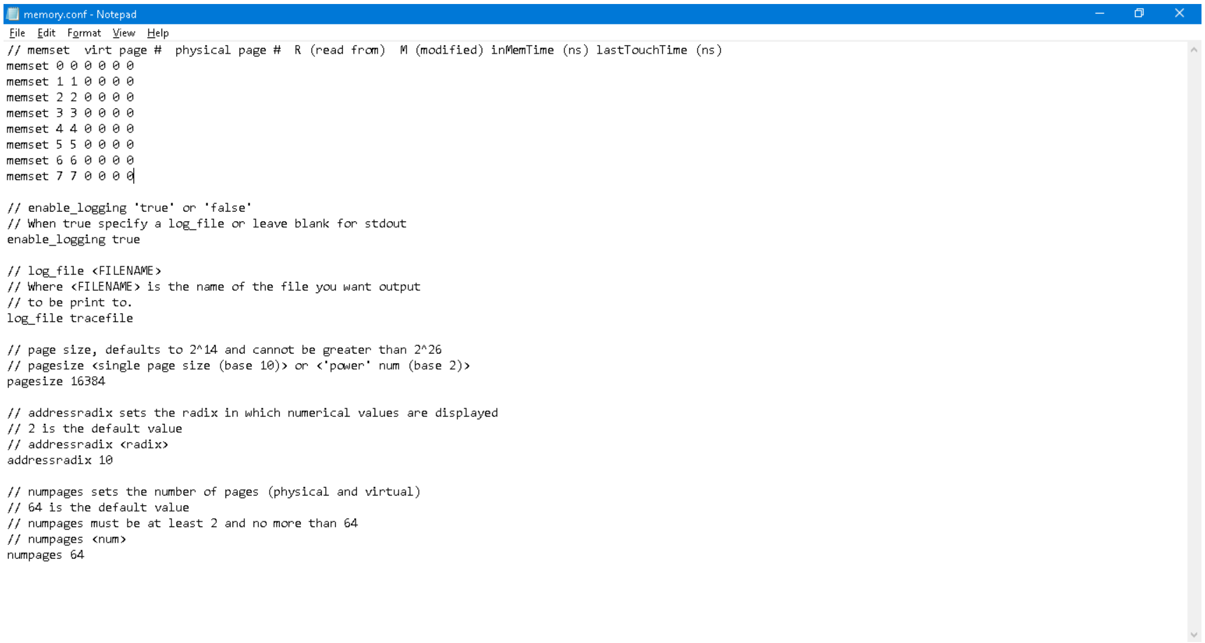
Create a command file that maps any 8 pages of physical memory to the first 8 pages of virtual memory, and then reads from one virtual memory address on each of the 64 virtual pages. Step through the simulator one operation at a time and see if you can predict which virtual memory addresses cause page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

As a background, I can say, the computer memories are created registers, where each register composes cell which able to store one-bit data, address lines used for data routing which means data buses. The matter of this is 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 easiest 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 1st bit of another one. However, the processes come and go, hence when we free a memory space is left. Fortunately, the device called Memory Management Unit keeps track of it and allocates the free memory space. If we want to access to the memory which has no physical address mapped, then a page fault warning is returned. But the processes differ in sizes. This issue is partially handled by participating the memory into segments called pages. And in our task, we are dealing with the static partitioning, where the size of each partition is pre-set. The “page size” variable in the memory configuration file of our program has been set to the default – 16384 and the number of pages is 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

// addressradix sets the radix in which numerical values are displayed
// 2 is the default value
// addressradix <radix>
addressradix 16

// 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
```

The next step was to setup the memory.conf. file to specify the initial content of the virtual memory map, specifically to map any 8 pages of physical memory to the 1st 8 pages of virtual memory.



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

// enable_logging 'true' or 'false'
// When true specify a log_file or leave blank for stdout
enable_logging true

// log_file <FILENAME>
// Where <FILENAME> is the name of the file you want output
// to be print to.
log_file tracefile

// 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

// addressradix sets the radix in which numerical values are displayed
// 2 is the default value
// addressradix <radix>
addressradix 10

// 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
```

memset – performs mapping between virtual and physical page;

enable_logging – True or False turn on/off logs;

log_file – path to log file and name;

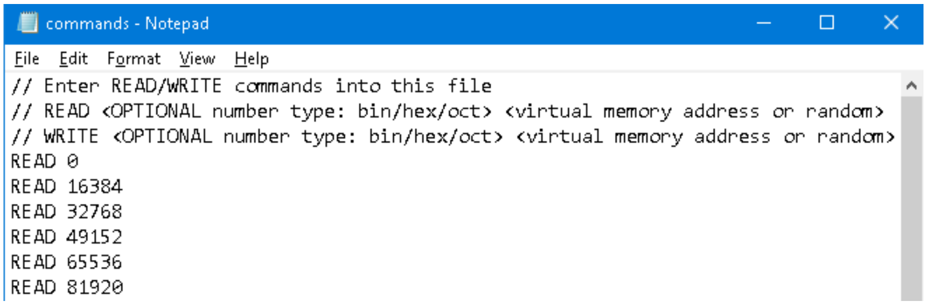
pagesize – page size, by default 2^14 and can not be > than 2^26;

addressradix – sets the radix in which numerical values are displaced.

Apart from that, I changed the radix in which numerical values are displayed to 10, so that it is decimal. I left the rest of the configuration information unchanged.

Configuration of the Commands file

Here, was needed to setup the commands file to specify a sequence of memory instructions to be performed, for reading from one virtual memory address on each of the 64 virtual pages. As the page size attribute is set to 16384 in the configuration file as default, so I decided to keep in that way. Therefore, to read from 1 address on each of the 64 pages, I decided to read from every 1 address on each page. Starting from 0-64 I defined READ operations on each multiple of 16384 in the following way.

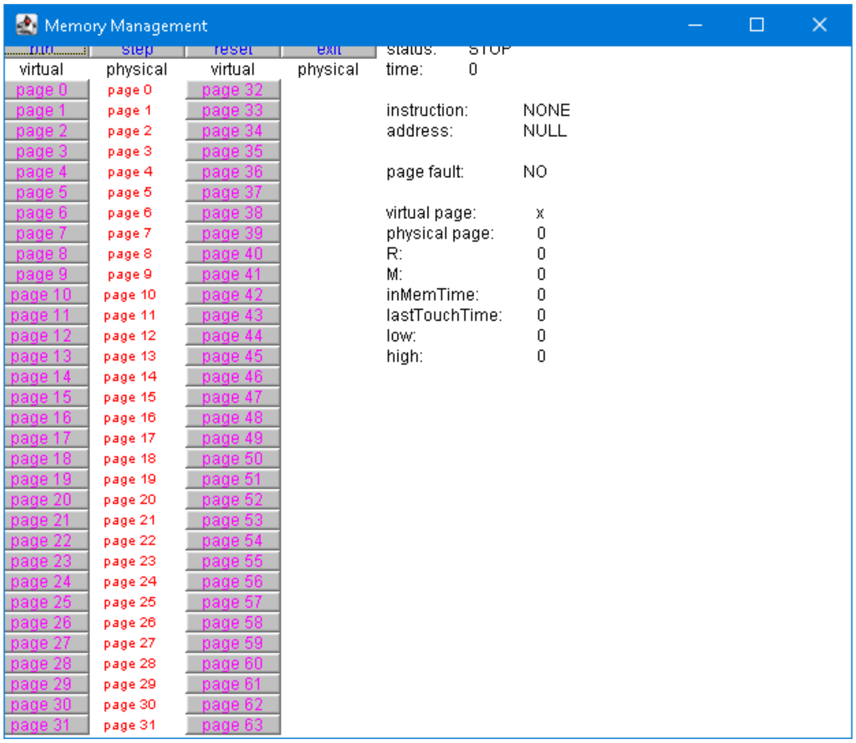


```
// Enter READ/WRITE commands into this file
// READ <OPTIONAL number type: bin/hex/oct> <virtual memory address or random>
// WRITE <OPTIONAL number type: bin/hex/oct> <virtual memory address or random>
READ 0
READ 16384
READ 32768
READ 49152
READ 65536
READ 81920
```

....

```
READ 884736
READ 901120
READ 917504
READ 933888
READ 950272
READ 966656
READ 983040
READ 999424
READ 1015808
```

Simulation the initial state

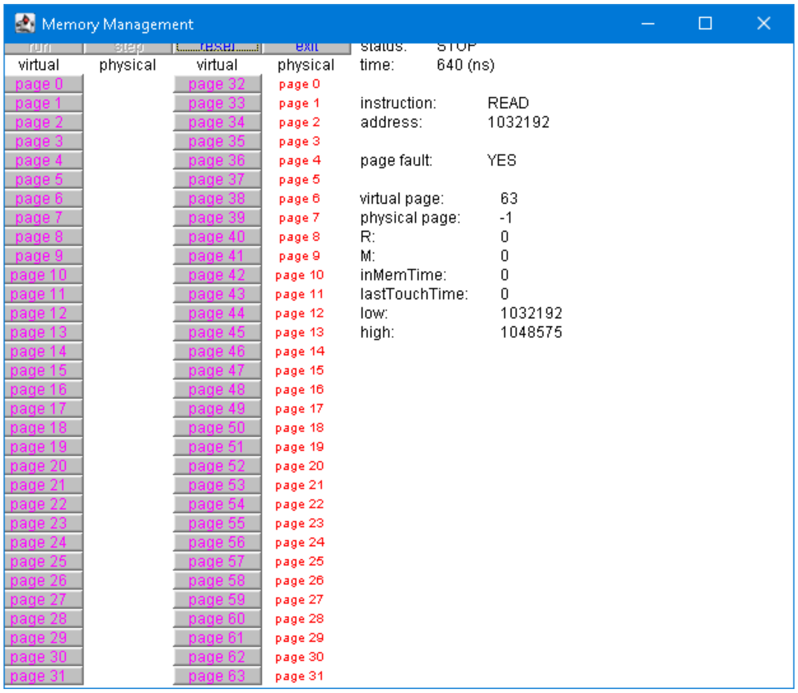


virtual	physical	virtual	physical
page 0	page 0	page 32	
page 1	page 1	page 33	
page 2	page 2	page 34	
page 3	page 3	page 35	
page 4	page 4	page 36	
page 5	page 5	page 37	
page 6	page 6	page 38	
page 7	page 7	page 39	
page 8	page 8	page 40	
page 9	page 9	page 41	
page 10	page 10	page 42	
page 11	page 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	

```
status:        STOP
time:          0

instruction:        NONE
address:            NULL

page fault:         NO

virtual page:       x
physical page:      0
R:                  0
M:                  0
inMemTime:          0
lastTouchTime:      0
low:                0
high:               0
```

After running the simulator with CMD, I see the initial state of the memory pages mapping. The virtual pages 0-7 are mapped to the physical pages that we specified in the configuration file. The virtual pages 8-31 are mapped to physical pages with the same number. Pages 32-63 have no mapping.

What is interesting, at the 32 virtual page fault appeared, however it was easy to predict, because this virtual page has no mapping to a physical page. Page fault occurs in which there has been a reference to a page which does not have mapping to a physical page. After the failed attempt to read virtual page 32, the simulator maps physical page 1 to it:



virtual	physical	virtual	physical
page 0		page 32	page 0
page 1		page 33	page 1
page 2		page 34	page 2
page 3		page 35	page 3
page 4		page 36	page 4
page 5		page 37	page 5
page 6		page 38	page 6
page 7		page 39	page 7
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:          1032192

page fault:       YES

virtual page:       63
physical page:      -1
R:                  0
M:                  0
inMemTime:          0
lastTouchTime:      0
low:                1032192
high:               1048575
```

After this simulation, we understood that each of the virtual pages 32-63 has been mapped to a physical page in the same order as pages 0-31. So, this indicate that ***First In – First Out*** (**FIFO**) algorithm was used.

Page Replacement Algorithm

In an operating system that uses paging for memory management, a ***page replacement algorithm*** is needed to decide which page needs to be replaced when new page comes in. Page Replacement Algorithm is an algorithm which defines what pages should be written to the disk when the new page needs to be distributed. This algorithm used in the simulation is FIFO. In this replacement algorithm, all the pages in the memory are kept in a queue. As the name suggests, those pages that were least recently used, are at the top of the queue, so for instance, after the failed attempt to read memory from virtual page 32, the physical page 1 is assigned which was previously assigned to the virtual page 0 (as a first one to be assigned in the simulation). So, the physical page was the first in and now when the system needs to assign a physical page to a virtual page it is the first one out.

However, the disadvantage of the FIFO algorithm is low efficiency because it does not consider which pages are frequently used and which are used once. For example, much better page replacement algorithm would be the Least Recently Used (LRU) memory management algorithm or the priority list.

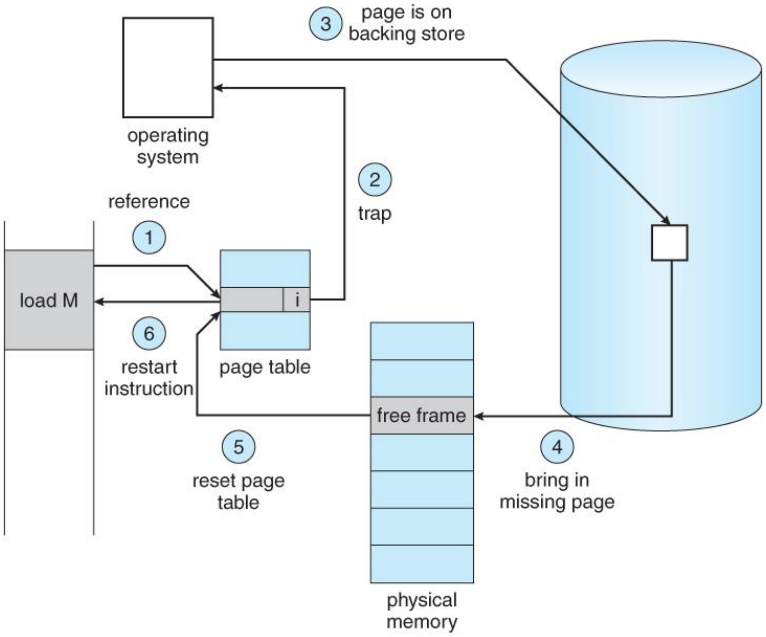
Page Fault

Page fault happens when the running program accesses the memory page that has been mapped into the virtual address space, but not still loaded in physical memory. And since, actual physical memory is smaller than virtual memory, page faults happen.

In the given picture below, there are the number of steps in handling Page Fault:

1) When the 1st checked is memory address requested, just to make sure that it was a valid memory request;

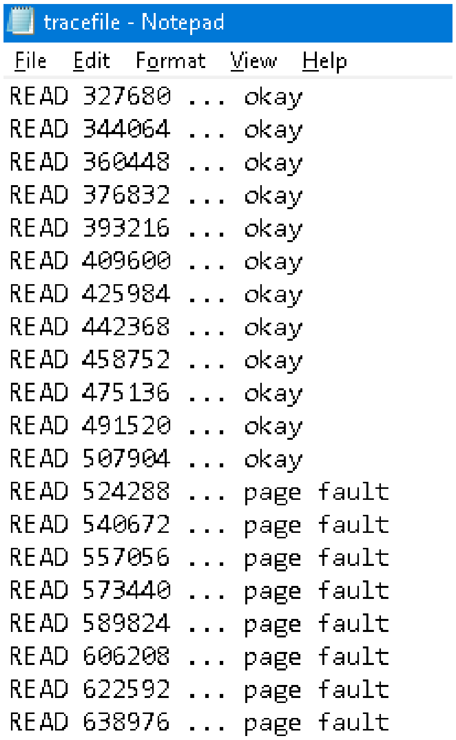
2) If the reference was invalid, the process is finished, in other words, the page must be paged in;
3) A free frame is located, possibly from the free frame list;
4) A disk operation is scheduled to bring in the necessary page from disk;
5) When the I/O operation is finish, the process's page table is updated with the new frame number and the invalid bit is changed that this is now a valid page reference;
6) The instruction which caused the page fault must restart from the beginning.



Source: Operating Systems: Virtual Memory (uic.edu)

Tracefile

The "tracefile" file contains a log of the operators since the simulation started together with the statuses. Just as a confirmation of the simulation process we can see that the 1st 32 READ operations were successful and all the remaining operations statuses indicate page fault:



```
READ 327680 ... okay
READ 344064 ... okay
READ 360448 ... okay
READ 376832 ... okay
READ 393216 ... okay
READ 409600 ... okay
READ 425984 ... okay
READ 442368 ... 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 589824 ... page fault
READ 606208 ... page fault
READ 622592 ... page fault
READ 638976 ... page fault
```