# EOPSY
# LAB 4

Filip Korzeniewski 293070

--[ YOUR TASK ]----------

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.

---

**memory.conf**
In memory.conf file it was set to map 8 pages of physical memory to the first 8 pages of virtual memory. Consequently physical pages with numbers 0, 1, 2, 3, 4, 5, 6, 7 were mapped to first 8 virtual pages (indexes from 0 to 7).

```
// 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
```

**commands:**

According to the memory.conf command file was set to make possible read all virtual memory addresses (64 pages). Page size was left default (16384) so the consequent

```
// 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>
// 0
READ 0
// 1
READ 16384
// 2
READ 32768
// 3
READ 49152
// 4
READ 65536
// 5
READ 81920
// 6
READ 98304
// 7
READ 114688
// 8
READ 131072
// 9
READ 147456
// 10
READ 163840
// 11
READ 180224
// 12
READ 196608
// 13
READ 212992
// 14
READ 229376
```

```
// 15
READ 245760
// 16
READ 262144
// 17
READ 278528
// 18
READ 294912
// 19
READ 311296
// 20
READ 327680
// 21
READ 344064
// 22
READ 360448
// 23
READ 376832
// 24
READ 393216
// 25
READ 409600
// 26
READ 425984
// 27
READ 442368
// 28
READ 458752
// 29
READ 475136
// 30
READ 491520
// 31
READ 507904
// 32
READ 524288
// 33
READ 540672
// 34
READ 557056
// 35
READ 573440
// 36
READ 589824
// 37
READ 606208
// 38
READ 622592
// 39
READ 638976
// 40
READ 655360
// 41
READ 671744
// 42
READ 688128
// 43
READ 704512
```

```
// 44
READ 720896
// 45
READ 737280
// 46
READ 753664
// 47
READ 770048
// 48
READ 786432
// 49
READ 802816
// 50
READ 819200
// 51
READ 835584
// 52
READ 851968
// 53
READ 868352
// 54
READ 884736
// 55
READ 901120
// 56
READ 917504
// 57
READ 933888
// 58
READ 950272
// 59
READ 966656
// 60
READ 983040
// 61
READ 999424
// 62
READ 1015808
// 63
READ 1032192
```

**Tracefile**

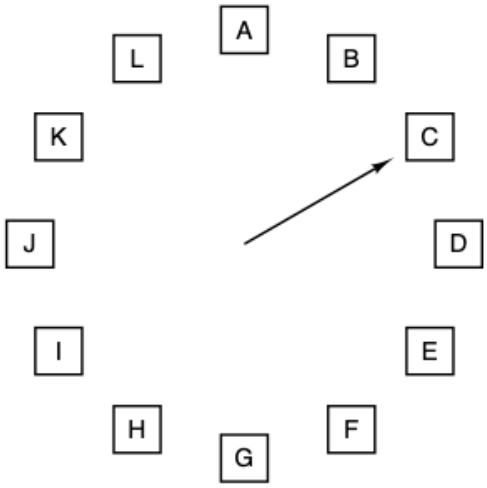
In the output file we can see that for pages 0-31 (first 32 pages) simulation was succeeded (no page fault). It is related to the memory configuration were 64 pages were set in total (32 virtual and 32 physical). Right after all 32 pages of physical memory (pages frames) had been mapped, the program tried to access the absent memory page, so the page fault was raised.

```
READ 0 ... okay
READ 16384 ... okay
READ 32768 ... okay
READ 49152 ... okay
READ 65536 ... okay
READ 81920 ... okay
READ 98304 ... okay
READ 114688 ... okay
READ 131072 ... okay
READ 147456 ... okay
```

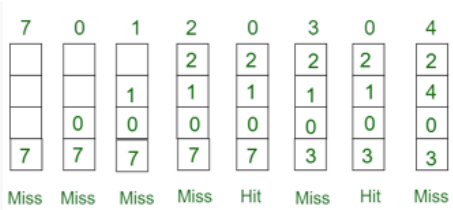
```
READ 163840 ... okay
READ 180224 ... okay
READ 196608 ... okay
READ 212992 ... okay
READ 229376 ... okay
READ 245760 ... okay
READ 262144 ... okay
READ 278528 ... okay
READ 294912 ... okay
READ 311296 ... okay
READ 327680 ... okay
READ 344064 ... okay
READ 360448 ... okay
READ 376832 ... okay
READ 393216 ... okay
READ 409600 ... okay
READ 425984 ... okay
READ 442368 ... okay
READ 458752 ... okay
READ 475136 ... okay
READ 491520 ... okay
READ 507904 ... okay ←32nd page (31st index)
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
READ 655360 ... page fault
READ 671744 ... page fault
READ 688128 ... page fault
READ 704512 ... page fault
READ 720896 ... page fault
READ 737280 ... page fault
READ 753664 ... page fault
READ 770048 ... page fault
READ 786432 ... page fault
READ 802816 ... page fault
READ 819200 ... page fault
READ 835584 ... page fault
READ 851968 ... page fault
READ 868352 ... page fault
READ 884736 ... page fault
READ 901120 ... page fault
READ 917504 ... page fault
READ 933888 ... page fault
READ 950272 ... page fault
READ 966656 ... page fault
READ 983040 ... page fault
READ 999424 ... page fault
READ 1015808 ... page fault
READ 1032192 ... page fault
```

**Conclusion:**
Firstly we can see that 8 pages were mapped as it was set. Then rest physical pages (24 pages) were mapped in default way. For the rest virtual pages page faults occurred. Due to this fact, the replacement had to be done. It can be seen that replacement algorithm that was used was FIFO. After 32 pages were mapped, then the first paged mapped (0 index, the oldest) was remapped to the 33rd page (32 index). Then second to 34th and so on.
In the FIFO algorithm, operating system keeps track of all pages in the memory in a queue and the oldest page is in front of the queue. When a page needs to be replaced, page in the front of the queue is selected for removal.

**Other algorithms:**
We can distinguish other page replacement algorithms. The FIFO algorithm is the basic one and was already mentioned in conclusion.
In the Clock replacement algorithm, when a a page fault occurs, the page the hand is pointing is inspected. The action depends on the R bit: if R==0, evict the page; if R==1, clear R and advance hand.




In the LRU (least recently used) algorithm, the least recently page is replaced. So it is checked which page is the oldest not used one.




One can see that in above example, when 3 comes, it takes place of 7 (like FIFO – no other page was used, so 7 is the oldest), then the 0 comes – no page fault. Then 4 comes – 0 was used recently, before there was 3, so ow it's a time for 1 to be replaced. This example demonstrates the algorithm in pretty good way.
There are also other algorithms like optimal solution, NRU (not recently used) algorithm, the second chance algorithm (improved FIFO).