EOPSY Lab 4

Memory Management

Jakub Szumski 295432

Performing Simulation

In order to perform the task we must run the simulation that is constructed with the use of Java language. Unpack the file called task 4 and in directory ftp execute ./make setup. This script creates directory work where you have to type commands: make compile and then make run to see the simulation window.

Mapping virtual pages to physical pages

In order to configure our mapping we have to edit file called *memory.conf*. Our task is to map any 8 pages of physical memory to the first 8 pages of virtual memory.

memory.conf

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

enable_logging true

log_file tracefile

pagesize 16384

addressradix 10

numpages 64
```

As presented above we assign our virtual pages to the physical pages according to the scheme below:

Virtual Page	Physical Page
0	7
1	6
2	5
3	4
4	3
5	2
6	1
7	0

Reading virtual pages

Next we have to configure file commands which will specify read operation for every 64 of virtual pages. As written in *memory.conf* file we've established that pagesize is 16384 addresses. Thus to ensure that address form each distinct virtual page. I've decided to use 1st address form ith page.

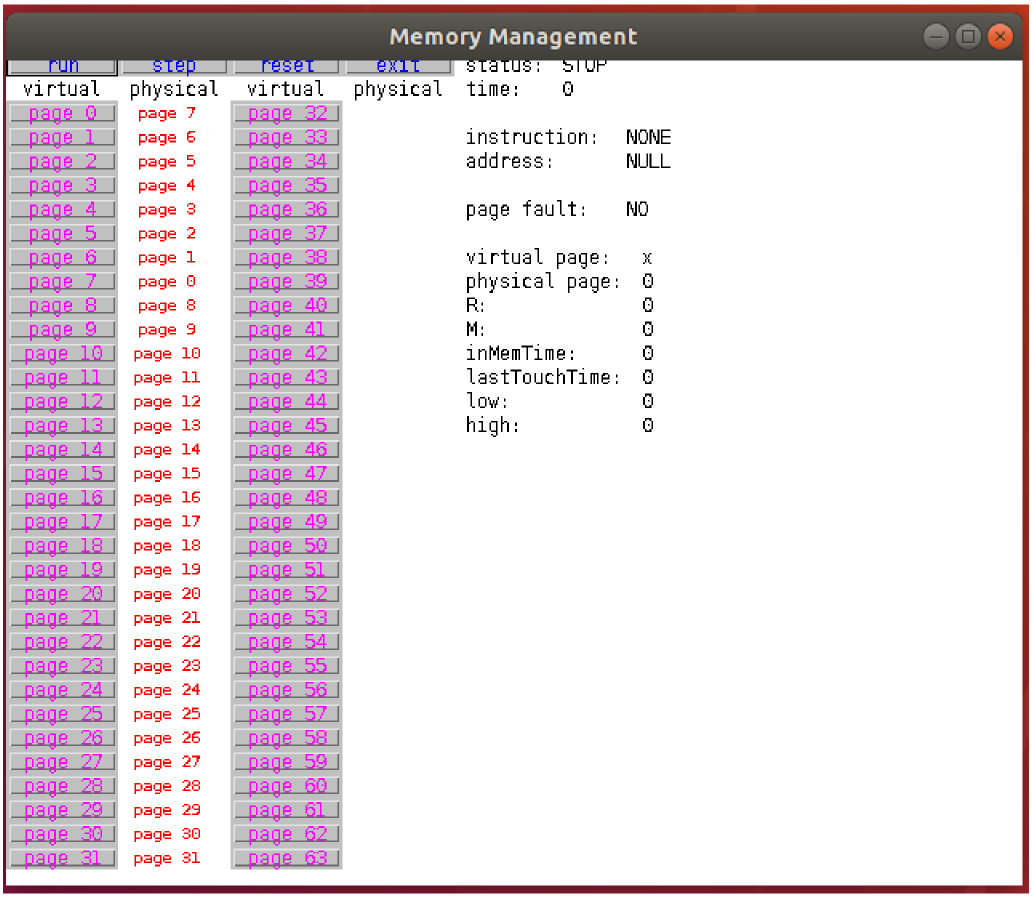
commands
READ 0 READ 16384 READ 32768 READ 49152 READ 65536 READ 81920 READ 98304 READ 114688 READ 131072 READ 147456 READ 163840 READ 180224 READ 196608 READ 212992 READ 229376 READ 245760 READ 262144 READ 278528 READ 294912 READ 311296 READ 327680 READ 344064 READ 360448 READ 376832 READ 393216 READ 409600 READ 425984 READ 442368 READ 458752 READ 475136 READ 491520 READ 507904 READ 524288 READ 540672 READ 557056 READ 573440 READ 589824 READ 606208 READ 622592 READ 638976 READ 655360 READ 671744 READ 688128 READ 704512 READ 720896 READ 737280 READ 753664 READ 770048 READ 786432 READ 802816 READ 819200 READ 835584 READ 851968

```
READ 868352
READ 884736
READ 901120
READ 917504
READ 933888
READ 950272
READ 966656
READ 983040
READ 999424
READ 1015808
READ 1032192
```

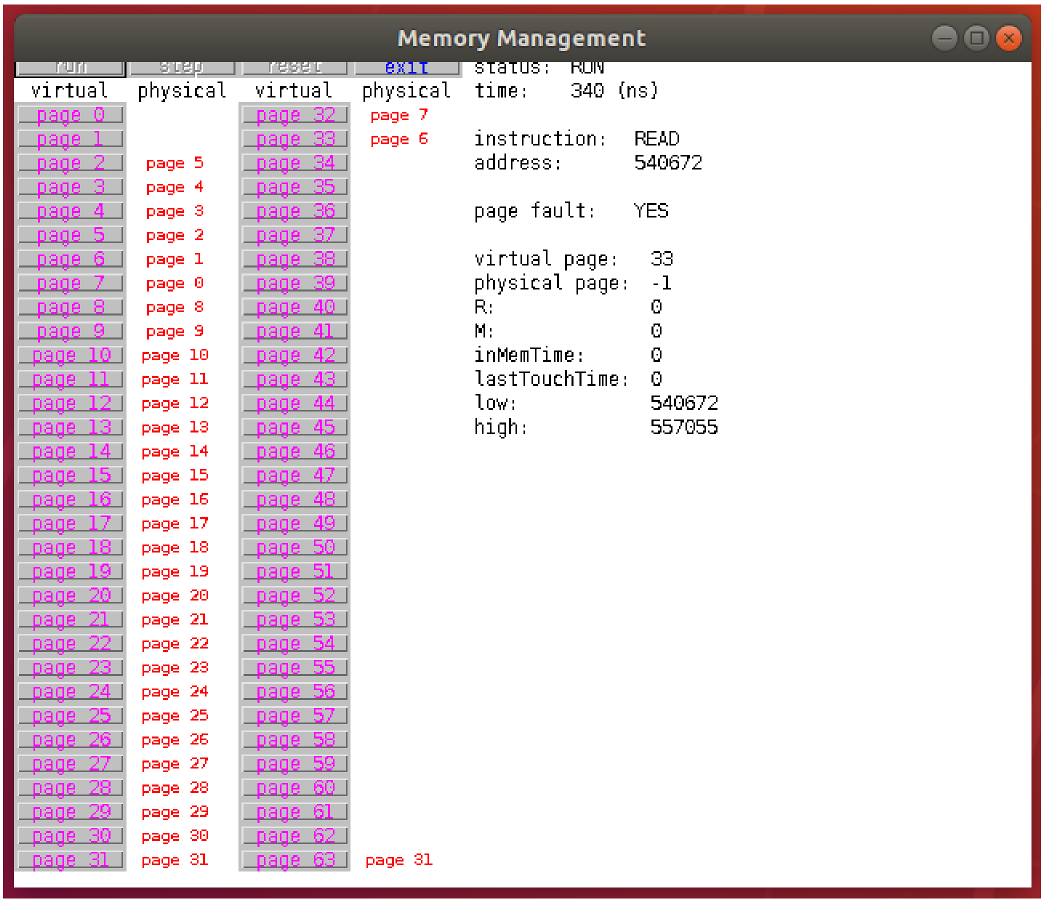
Page faults predictions

Now we will run our simulation and try to predict which READ operation will trigger page fault. Page fault is a situation where virtual page is not mapped to any physical page and we try to read address from this virtual page. In such situation proper physical page will be mapped to virtual page that informed about page fault.

After typing make run command we can see simulation window appear:



As visible on the screenshot first 8 virtual pages have been have been assigned physical pages. As for the rest mapping have been performed by default. Also we can observe that for pages 32-63 there is no physical page assigned. Therefore I predict that any attempt of accessing this pages will result in a *page fault.* To verify this assumption I will start this simulation and see if my assumption is correct.



As predicted any attempt of accessing the mentioned virtual pages results in page fault. We can also notice that physical page assigned to those values is equal to -1.

Another indication that this assumption is correct may be found in *tracefile* file. Let's see it's contents.

tracefile
1) READ 0 ... okay
2) READ 16384 ... okay
3) READ 32768 ... okay
4) READ 49152 ... okay
5) READ 65536 ... okay
6) READ 81920 ... okay
7) READ 98304 ... okay
8) READ 114688 ... okay
9) READ 131072 ... okay
10) READ 147456 ... okay
11) READ 163840 ... okay
12) READ 180224 ... okay
13) READ 196608 ... okay
14) READ 212992 ... okay
15) READ 229376 ... okay
16) READ 245760 ... okay
17) READ 262144 ... okay
18) READ 278528 ... okay
19) READ 294912 ... okay
20) READ 311296 ... okay
21) READ 327680 ... okay
22) READ 344064 ... okay
23) READ 360448 ... okay
24) READ 376832 ... okay
25) READ 393216 ... okay
26) READ 409600 ... okay

```
    27) READ 425984 ... okay
    28) READ 442368 ... okay
    29) READ 458752 ... okay
    30) READ 475136 ... okay
    31) READ 491520 ... okay
    32) READ 507904 ... okay
    33) READ 524288 ... page fault
    34) READ 540672 ... page fault
    35) READ 557056 ... page fault
    36) READ 573440 ... page fault
    37) READ 589824 ... page fault
    38) READ 606208 ... page fault
    39) READ 622592 ... page fault
    40) READ 638976 ... page fault
    41) READ 655360 ... page fault
    42) READ 671744 ... page fault
    43) READ 688128 ... page fault
    44) READ 704512 ... page fault
    45) READ 720896 ... page fault
    46) READ 737280 ... page fault
    47) READ 753664 ... page fault
    48) READ 770048 ... page fault
    49) READ 786432 ... page fault
    50) READ 802816 ... page fault
    51) READ 819200 ... page fault
    52) READ 835584 ... page fault
    53) READ 851968 ... page fault
    54) READ 868352 ... page fault
    55) READ 884736 ... page fault
    56) READ 901120 ... page fault
    57) READ 917504 ... page fault
    58) READ 933888 ... page fault
    59) READ 950272 ... page fault
    60) READ 966656 ... page fault
    61) READ 983040 ... page fault
    62) READ 999424 ... page fault
    63) READ 1015808 ... page fault
    64) READ 1032192 ... page fault
```

As we can see for last 32 pages (32-63) there is a page fault error. This concludes our prediction.

Page Replacement Algorithm

Our page replacement algorithm can be found in *./tast4/work/PageFault.java* file. To be exact the specific function is called:

```
public static void replacePage(Vector mem, int virtPageNum, int replacePageNum, ControlPanel controlPanel)
```

In the documentation we read that implemented algorithm is *FIFO(First-In First-Out).*

Description: Operating system holds a list of all pages currently in memory, with the most recent arrival at the tail and the least recent arrival at the head. On a page fault, the page at the head is removed and the new page is added to the tail of the list.

If we take a look at the end of the simulation we can observe that the first mapping removed was the first one that occurred.