17.05.2021

Jan Jakubiak

289834

# EOPSY

## Lab 4 – Memory Management

Memory management, which is the focal point of this laboratory, is a process done by the operating system. It dynamically allocates parts of its memory to programs and processes, freeing it and allocating to different processes when it's no longer needed.

One of the methods used in memory management, and the one we will be trying out in this laboratory is called paging. In this method the computer uses the secondary, hard-drive memory to store data in regularly sized parts called pages. Paging allows programs to use more memory than physically available in the computer, and as such is used regularly.

Task 4 starts with us having to manually map the 8 first pages of virtual memory to any 8 pages of physical memory we wanted. I mapped them in a following way:

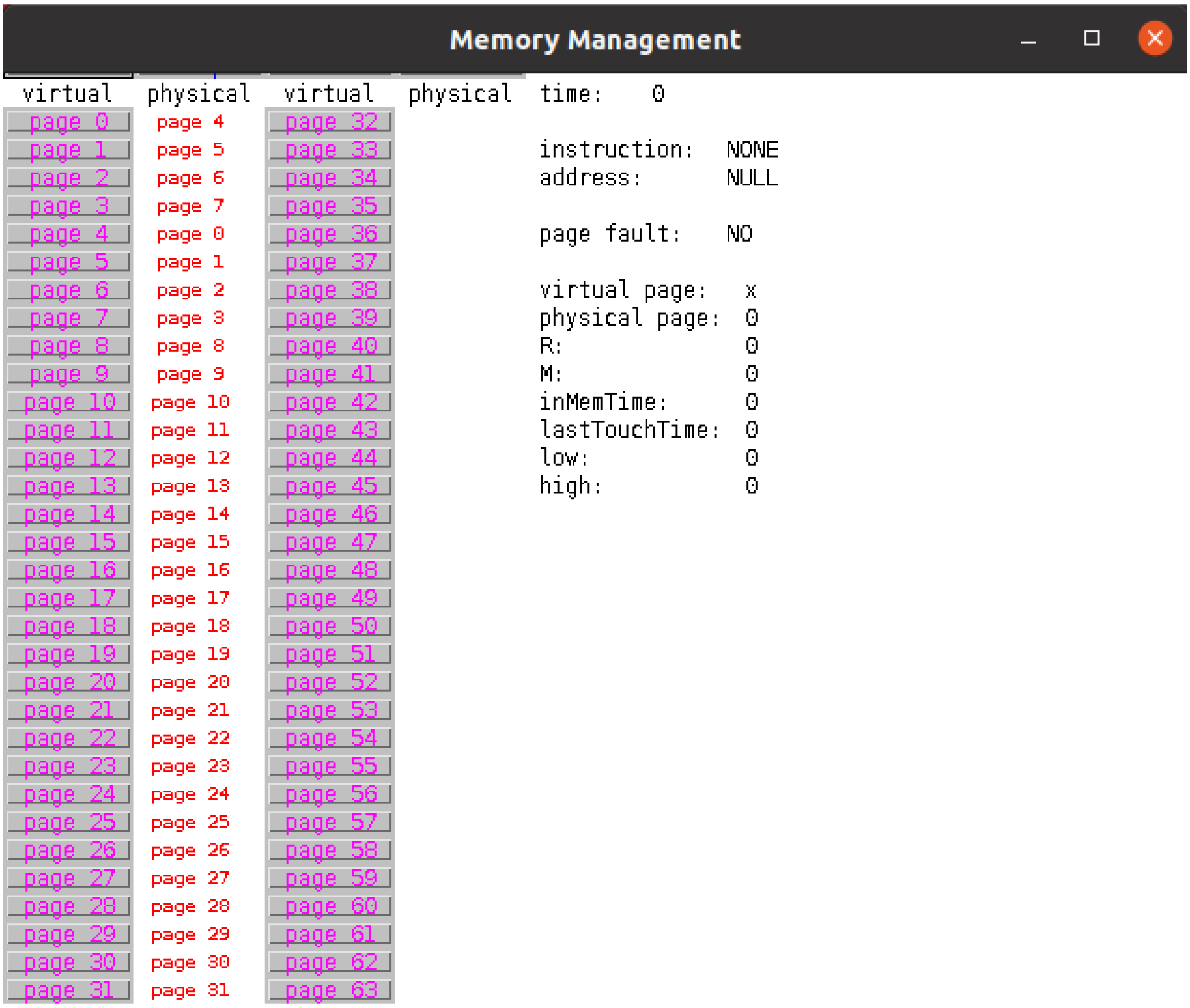
memory.conf
~/Desktop/task4/work

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

We then had to configure the commands file to perform the “read” operation from one address for every virtual page. Having set the pagesize to 16384 addresses, the addresses used were equal to 16384 * *i* , where *i* is the page number – from 0 to 63. Attempting to read from an address from every virtual page allows us to check if all the pages are properly mapped and usable.

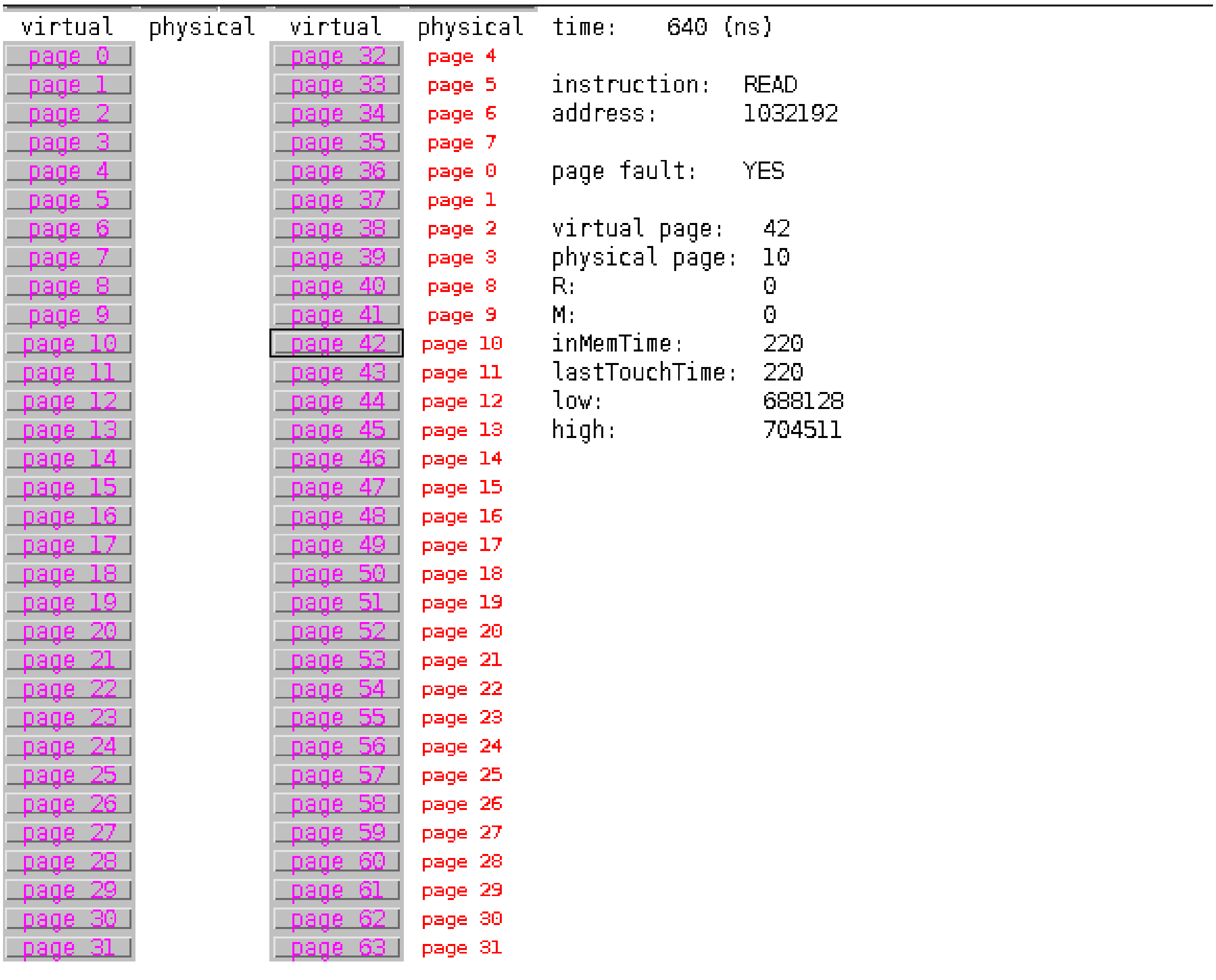
```
READ 0
READ 16384
READ 32768
READ 49152
READ 65536
READ 81920
READ 98304
READ 114688
READ 131072
READ 147456
READ 163840
READ 180224
READ 196608
READ 212992
READ 229376
READ 245760
READ 262144
READ 278528
READ 294912
READ 311296
READ 327680
READ 344064
READ 360448
READ 376832
READ 393216
READ 409600
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
```

This was the simulation program before running the simulation. We can see the first 8 pages mapped in a way we specified in the memory.conf file, and the next pages up to 32nd are mapped by default – each to its corresponding virtual page.




Since the virtual pages from 32 upwards are not mapped to any physical pages, we can expect a page fault when trying to access them. A page fault is an error occurred when trying to reference a page that is not mapped – exactly what we will be attempting when running the simulation.

After running the simulation and going through all 64 pages, we can see the following output from the program:




We can see that the pages from 0 to 31 are now not mapped, while all the higher pages are. That's the result of the page fault, which was met here – as can be seen on the screenshot above (page fault: YES). When the page fault occurs, the page replacement algorithm decides which physical page to map the troublesome virtual page to. Output from the tracefile confirms these observations, and shows that trying to access the pages from 32 upwards caused a page fault in each case:

```
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
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
```

The last thing we need to do is to check what page replacement algorithm was used and how it works. The file PageFault.java has the algorithm and its description inside.

```java
public class PageFault {

  /**
   * The page replacement algorithm for the memory management sumulator.
   * This method gets called whenever a page needs to be replaced.
   * <p>
   * The page replacement algorithm included with the simulator is
   * FIFO (first-in first-out).  A while or for loop should be used
   * to search through the current memory contents for a canidate
   * replacement page.  In the case of FIFO the while loop is used
   * to find the proper page while making sure that virtPageNum is
   * not exceeded.
```

The First In First Out is the simplest page replacement algorithm. The system keeps the pages from the memory in a queue, sorted according to their age, with the oldest page being in the front of the queue. When a page fault occurs, the page from the front of the queue is removed and remapped to the page causing the error. We can see it confirmed in the screenshots – after the simulation, pages 32-63 are all mapped to the physical pages in the same order the pages 0-32 were previously.

| virtual | physical | virtual |
|---|---|---|
| page 0 | page 4 | page 32 |
| page 1 | page 5 | page 33 |
| page 2 | page 6 | page 34 |
| page 3 | page 7 | page 35 |
| page 4 | page 0 | page 36 |
| page 5 | page 1 | page 37 |
| page 6 | page 2 | page 38 |
| page 7 | page 3 | 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 |

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