# [EOPSY] LAB 4

ERNEST POKROPEK

WARSAW UNIVERSITY OF TECHNOLOGY, 2020

# Table of contents

# 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.

# Simulation

Having the configuration file **memory.conf**, it was furtherly processed to map 8 pages of physical memory to first 8 pages of virtual memory. For this purpose, first 8 (from 0 to 7) pages of physical memory were used. The full contents of memory.conf file can be found in the *Appendix*.

After testing the code and performing a successful simulation, the configuration file for reading the pages has been created using Python script which generated READ commands for addresses of value (n+1)*16000 (where n is an integer from 0 to 63), assuming that each page's address ends in the interval of n*16000 and (n+1)*16000; both the script and result configuration file are to be found in the *Appendix*.

On the Figure 1, the first page fault has been documented, as for the virtual page 32. This is expected, as only 8 of the pages were mapped to the memory, whereas the remaining 23 were mapped automatically. Trying to execute "READ" further, one will notice that each next page is unmapped, and such procedure will result in another page fault. This has been presented on the Figure 2.

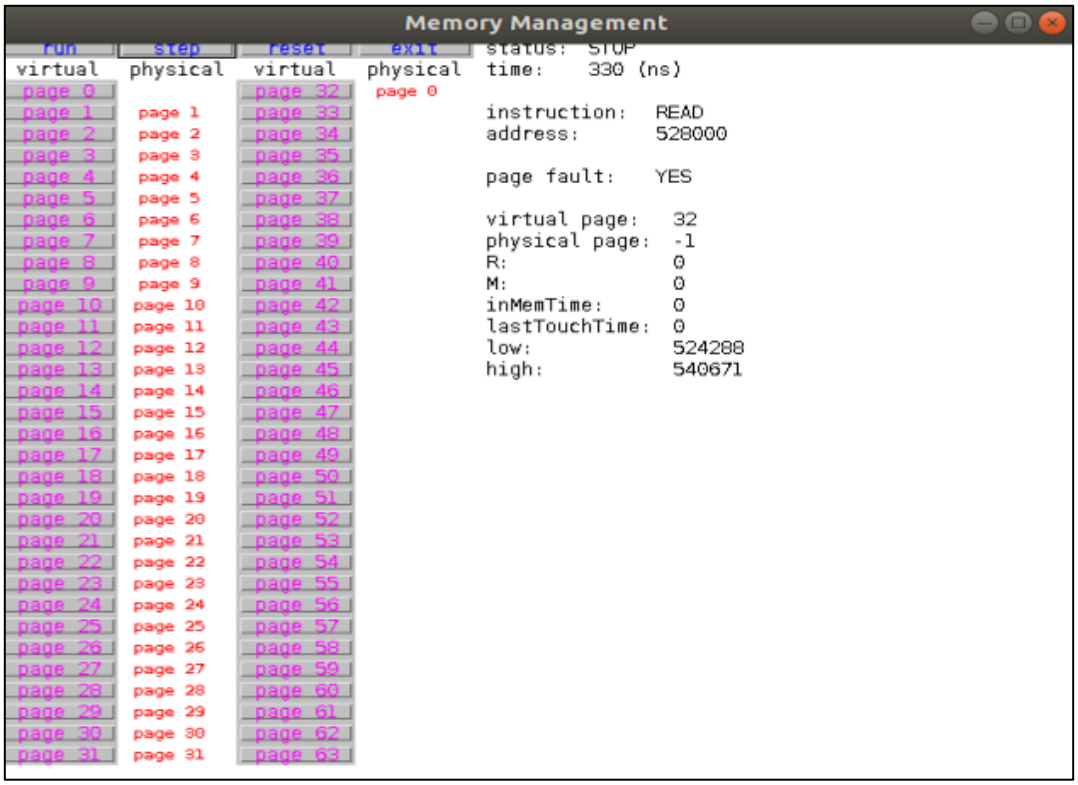


Figure 1. Memory management simulation on first page fault

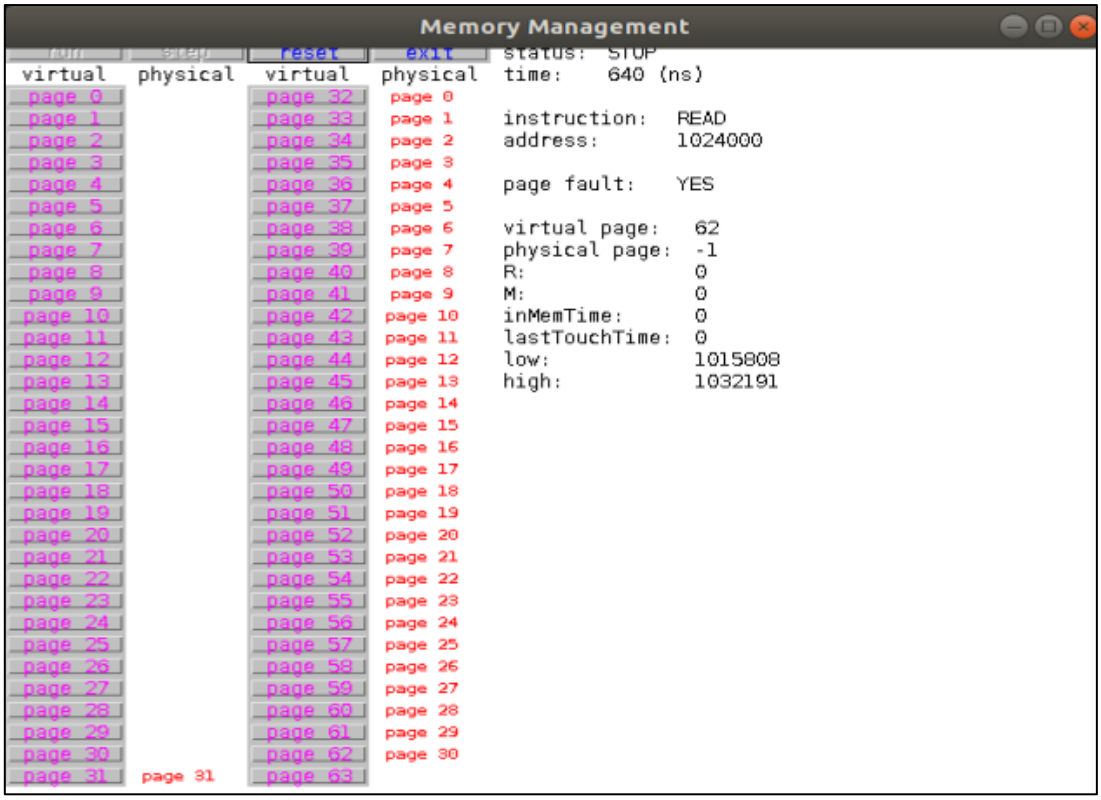


Figure 2. End of the simulation

The commands, as well as traceback, are presented in the Table 1. It is worthy to notice, that command READ 688000 has been executed properly – it is due to the address overlapping, i.e. same page was read twice. As it can be seen, the addresses in memory can be quite unpredictable, so the Python script's idea was not ideal. For the rest of the commands, the simulation went as expected (resulting in page fault for pages 32 and next).

| commands | tracefile |
|---|---|
| READ 16000 | READ 16000 ... okay |
| READ 32000 | READ 32000 ... okay |
| READ 48000 | READ 48000 ... okay |
| READ 64000 | READ 64000 ... okay |
| READ 80000 | READ 80000 ... okay |
| READ 96000 | READ 96000 ... okay |
| READ 112000 | READ 112000 ... okay |
| READ 128000 | READ 128000 ... okay |
| READ 144000 | READ 144000 ... okay |
| READ 160000 | READ 160000 ... okay |
| READ 176000 | READ 176000 ... okay |
| READ 192000 | READ 192000 ... okay |
| READ 208000 | READ 208000 ... okay |
| READ 224000 | READ 224000 ... okay |
| READ 240000 | READ 240000 ... okay |
| READ 256000 | READ 256000 ... okay |
| READ 272000 | READ 272000 ... okay |
| READ 288000 | READ 288000 ... okay |
| READ 304000 | READ 304000 ... okay |
| READ 320000 | READ 320000 ... okay |
| READ 336000 | READ 336000 ... okay |
| READ 352000 | READ 352000 ... okay |

| | |
|---|---|
| READ 368000 | READ 368000 ... okay |
| READ 384000 | READ 384000 ... okay |
| READ 400000 | READ 400000 ... okay |
| READ 416000 | READ 416000 ... okay |
| READ 432000 | READ 432000 ... okay |
| READ 448000 | READ 448000 ... okay |
| READ 464000 | READ 464000 ... okay |
| READ 480000 | READ 480000 ... okay |
| READ 496000 | READ 496000 ... okay |
| READ 512000 | READ 512000 ... okay |
| READ 528000 | READ 528000 ... page fault |
| READ 544000 | READ 544000 ... page fault |
| READ 560000 | READ 560000 ... page fault |
| READ 576000 | READ 576000 ... page fault |
| READ 592000 | READ 592000 ... page fault |
| READ 608000 | READ 608000 ... page fault |
| READ 624000 | READ 624000 ... page fault |
| READ 640000 | READ 640000 ... page fault |
| READ 656000 | READ 656000 ... page fault |
| READ 672000 | READ 672000 ... page fault |
| READ 688000 | READ 688000 ... okay |
| READ 704000 | READ 704000 ... page fault |
| READ 720000 | READ 720000 ... page fault |
| READ 736000 | READ 736000 ... page fault |
| READ 752000 | READ 752000 ... page fault |
| READ 768000 | READ 768000 ... page fault |
| READ 784000 | READ 784000 ... page fault |
| READ 800000 | READ 800000 ... page fault |
| READ 816000 | READ 816000 ... page fault |
| READ 832000 | READ 832000 ... page fault |
| READ 848000 | READ 848000 ... page fault |
| READ 864000 | READ 864000 ... page fault |
| READ 880000 | READ 880000 ... page fault |
| READ 896000 | READ 896000 ... page fault |
| READ 912000 | READ 912000 ... page fault |
| READ 928000 | READ 928000 ... page fault |
| READ 944000 | READ 944000 ... page fault |
| READ 960000 | READ 960000 ... page fault |
| READ 976000 | READ 976000 ... page fault |
| READ 992000 | READ 992000 ... page fault |
| READ 1008000 | READ 1008000 ... page fault |
| READ 1024000 | READ 1024000 ... page fault |

Table 1. Commands file contents and traceback

# Discussion

Taking a peek at the code of "PageDefault.java", we are able to analyse the code of function **replacePage** (available in the Appendix). It states, that "oldest" physical page that is mapped to a virtual page is then removed and assigned to the virtual page that haven't yet been mapped. This algorithm is called **First-In, First-Out**, FIFO for short. It works like a queue – the first thing that comes towards the FIFO, will be the first one to leave.

# Appendix

**memory.conf**

```
// 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
```

Java function to replace the page using FIFO algorithm:

```
 public static void replacePage ( Vector mem , int virtPageNum , int replace
PageNum , ControlPanel controlPanel )
  {
    int count = 0;
    int oldestPage = -1;
    int oldestTime = 0;
    int firstPage = -1;
    int map_count = 0;
    boolean mapped = false;

    while ( ! (mapped) || count != virtPageNum ) {
      Page page = ( Page ) mem.elementAt( count );
      if ( page.physical != -1 ) {
```

```
        if (firstPage == -1) {
          firstPage = count;
        }
        if (page.inMemTime > oldestTime) {
          oldestTime = page.inMemTime;
          oldestPage = count;
          mapped = true;
        }
      }
      count++;
      if ( count == virtPageNum ) {
        mapped = true;
      }
    }
    if (oldestPage == -1) {
      oldestPage = firstPage;
    }
    Page page = ( Page ) mem.elementAt( oldestPage );
    Page nextpage = ( Page ) mem.elementAt( replacePageNum );
    controlPanel.removePhysicalPage( oldestPage );
    nextpage.physical = page.physical;
    controlPanel.addPhysicalPage( nextpage.physical , replacePageNum );
    page.inMemTime = 0;
    page.lastTouchTime = 0;
    page.R = 0;
    page.M = 0;
    page.physical = -1;
  }
```

Script for generate the commands file:

```
with open('commands', mode='w') as f:
    for i in range(64):
        f.write('READ ' + str(int((i+1)*16e3)) + '\n')
```