Operating Systems

Laboratory 4

Author: Wiktor Łazarski

Field of study: Computer Science

Faculty: Electronics and Information Technology, Warsaw University of Technology

Memory Management

0. Laboratory performing script

To perform all researches I wrote a bash script called *mm_lab.sh*. There are three option to execute that script:

1. With '*setup*' parameter – this call results that the script will setup and compile simulator environment.
2. With '*run*' parameter – this call results that the script will run a simulator with a configuration files(*memory.conf, commands*).
3. With *'clean'* parameter – this call results that the script will remove all the obtain results and MOSS Simulator environment.

Link: https://github.com/wiktorlazarski/Operating-Systems/tree/master/4.memory%20management

1. Mapping virtual to physical pages

First we need to map any 8 pages of physical memory to the first 8 pages of virtual pages. To do this I wrote a *memory.conf* file in a following way.

./memory.conf

```
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
```

Below the description of each *memory.conf* file can be found in a table below.

Keyword	Description
memset	*memset* is a keyword that performs mapping between virtual page and physical page. Following digits represent as follow: virt page # physical page # R (read from) M (modified) inMemTime (ns)
enable_logging	*enable_logging* 'true' or 'false' When true specify a log_file or leave blank for stdout
log_file	*log_file* <FILENAME> Where <FILENAME> is the name of the file you want output to be print to.
pagesize	page size, defaults to 2^14 and cannot be greater than 2^26 pagesize <single page size (base 10)> or <'power' num (base 2)>
addressradix	*addressradix* sets the radix in which numerical values are displayed 2 is the default value addressradix <radix>
numpages	*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>

To fulfil the first part of laboratory task, 'map any 8 physical pages to first 8 virtual pages', *memset* keyword was used. This mapping process is highlighted above, by a red square shown in a table which presents *memory.conf* file. In this step I decided to map virtual pages to physical pages as follow:

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

2. Reading every virtual page

Then we need to specify the steps that the simulator will be performing. For this we will write a separate file called *commands*. In this file, according to the laboratory task, I need to specify read operation from every 64 virtual pages specified in my simulator(by keyword *numpages* in *memory.conf* file).

To ensure that every address in every virtual page will be read we need to do some math. In *memory.conf* file, we specified that every virtual page will consists of 16384 addresses via keyword *pagesize*. Therefore, to ensure that address from every virtual page will be read we need to write READ command in *commands* file that will read every address that is multiplication of 16384. We can pick any but I have decided to read every first address of every i^{th} virtual page.

Below, you can see full *commands* file. This file was generating using additional program written in C++ (code available in git repository *generate_commands.cpp*).

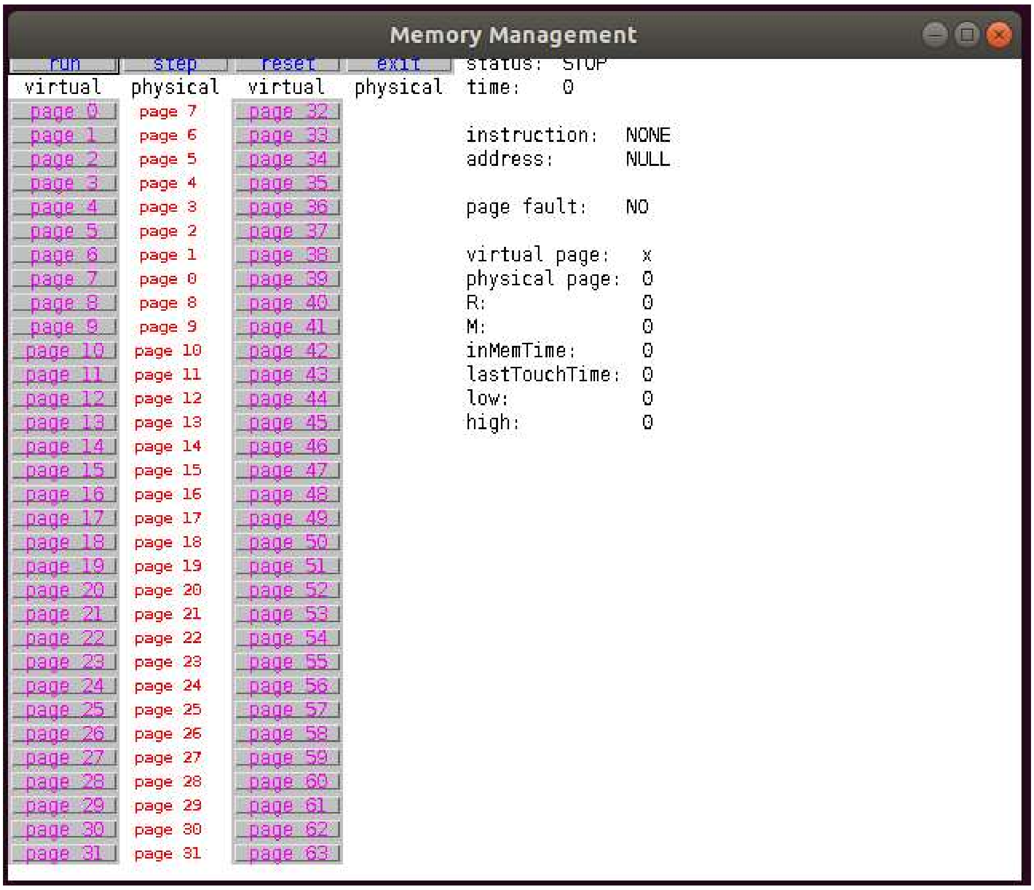
```
./commands
READ 0
READ 16384
READ 32768
READ 49152
READ 65536
READ 81920
READ 98304
READ 114688
READ 131072
READ 147456
READ 163840
READ 180224
READ 196608
READ 212992
READ 229376
READ 245760
READ 262144
READ 278528
READ 294912
READ 311296
READ 327680
READ 344064
READ 360448
READ 376832
READ 393216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

READ keyword is follow by a decimal value of an address to be read from virtual memory.

3. Predicting page faults

Now, we will run the simulator and we will try to predict which READ operation will cause *page fault*. *Page fault* is a situation where virtual page is not mapped to any physical page and we attempt to read address from this virtual page. When it occurs proper physical page will be mapped to virtual page that informed about *page fault*.

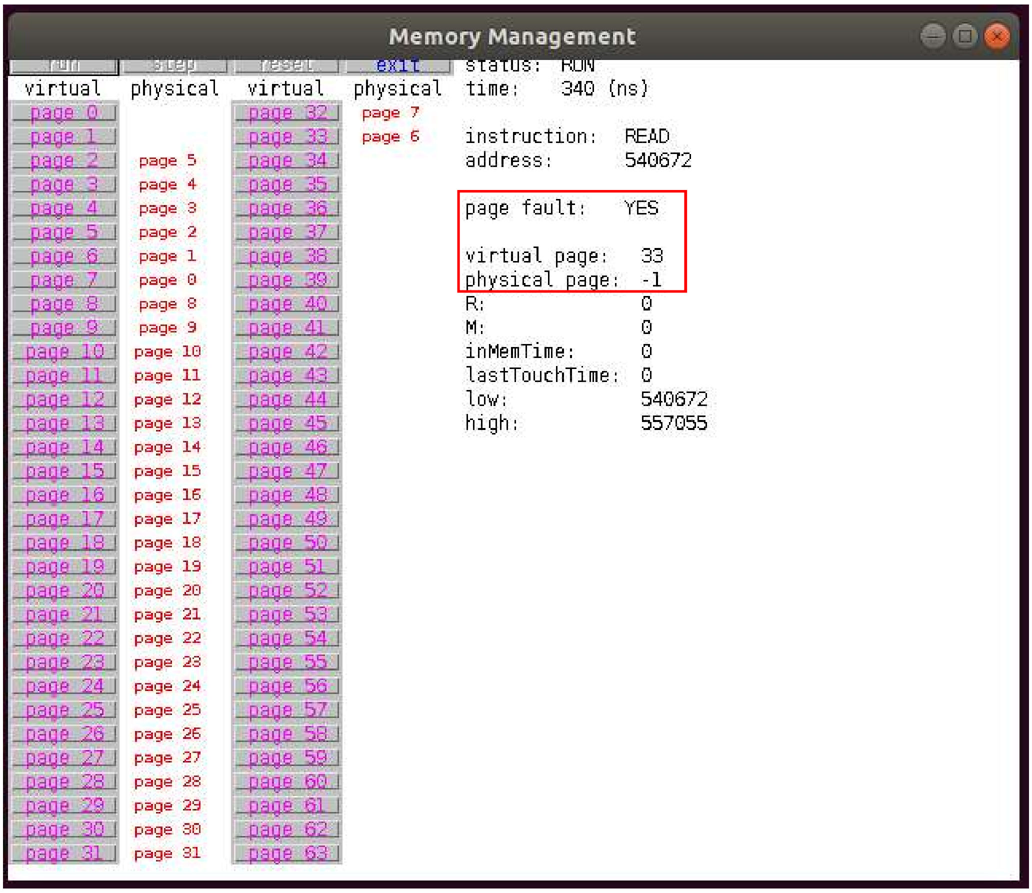
Starting the simulator the following window appears:



Remark: According to a *memory.conf* file mapping to first 8 virtual pages was performed. Others, visible in a simulator, mapping must have been performed by a simulator by default.

Looking at the simulator, to be specific how virtual memory is mapped to a physical one, we can see that every virtual page between 32-63 is not mapped to any virtual page. Therefore, I am sure that reading addresses, within ranges of those virtual pages, will cause *page fault*. I will run a simulator and see an information appearing on the left panel to check if I am correct.

Below, you may observe screenshot of a window during running and see that page faults start exactly as predicted, meaning from reading address in virtual page #32 and follows to the end of simulation, meaning to the last reading address(from virtual page #63). This information is highlighted by a red rectangle applied on simulation window.



Also, it can be noticed that my previous statement, about the fact that virtual pages which are not mapped to any physical page will cause page fault, is confirmed in this simulation because -1 assigned to a physical page indicates that virtual page is not assigned to any physical page.

Then *tracefile* file, which outputs information about if address was read correctly or caused page fault situation, can be looked at to indicate correctness of my predictions.

Table, containing output of a *tracefile* file, is present below.

./tracefile
0) READ 0 ... okay
1) READ 16384 ... okay
2) READ 32768 ... okay
3) READ 49152 ... okay
4) READ 65536 ... okay
5) READ 81920 ... okay
6) READ 98304 ... okay
7) READ 114688 ... okay
8) READ 131072 ... okay
9) READ 147456 ... okay
10) READ 163840 ... okay
11) READ 180224 ... okay
12) READ 196608 ... okay
13) READ 212992 ... okay
14) READ 229376 ... okay
15) READ 245760 ... okay
16) READ 262144 ... okay
17) READ 278528 ... okay
18) READ 294912 ... okay
19) READ 311296 ... okay
20) READ 327680 ... okay
21) READ 344064 ... okay
22) READ 360448 ... okay
23) READ 376832 ... okay
24) READ 393216 ... okay
25) READ 409600 ... okay
26) READ 425984 ... okay
27) READ 442368 ... okay
28) READ 458752 ... okay
29) READ 475136 ... okay
30) READ 491520 ... okay
31) READ 507904 ... okay
32) READ 524288 ... page fault
33) READ 540672 ... page fault
34) READ 557056 ... page fault
35) READ 573440 ... page fault
36) READ 589824 ... page fault
37) READ 606208 ... page fault
38) READ 622592 ... page fault
39) READ 638976 ... page fault
40) READ 655360 ... page fault
41) READ 671744 ... page fault
42) READ 688128 ... page fault
43) READ 704512 ... page fault
44) READ 720896 ... page fault
45) READ 737280 ... page fault
46) READ 753664 ... page fault
47) READ 770048 ... page fault
48) READ 786432 ... page fault
49) READ 802816 ... page fault
50) READ 819200 ... page fault
51) READ 835584 ... page fault
52) READ 851968 ... page fault
53) READ 868352 ... page fault
54) READ 884736 ... page fault
55) READ 901120 ... page fault
56) READ 917504 ... page fault
57) READ 933888 ... page fault
58) READ 950272 ... page fault
59) READ 966656 ... page fault
60) READ 983040 ... page fault
61) READ 999424 ... page fault
62) READ 1015808 ... page fault
63) READ 1032192 ... page fault

Remark: As predicted, reading first 32(0-31) virtual pages resulted in correct address reading and reading last 32(32-63) virtual pages resulted in page fault situation.

4. Locating in the sources page replacement algorithm

Implementation of a page replacement algorithm can be found in *./task4/work/PageFault.java* file. To be more specific function in this class is called:

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

From a documentation comment we can read that algorithm implemented is: *FIFO(First-In First-Out)*.

Algorithm Description: The operating system maintains 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.

Simulation confirms usage of this algorithm because when we will look at the step when page fault occurs. The first mapping that is removed was the first one that occurred. Below I present memory mapping after ending simulation.