Urszula Chmielewska

300167

EOPSY

Lab 4

Memory Management

TASK

The goal of this task was to configure a command file which maps any 8 pages of physical memory to the first 8 pages of virtual memory. After that it reads from one virtual memory address on each of the 64 virtual pages.

INTRODUCTION

Memory Management is functionality of an operating system to manages primary memory and moves processes between main memory and disk during execution. It has knowledge about each memory localization, no matter if it is assigned to any process or stays free. It also checks how much memory has to be allocated for the process.

While loading and removing processes from the memory, the free memory space is divided into smaller pieces. After some time, memory blocks are so small that there is not enough memory space for the process to be allocated. Such problem is known as **fragmentation**.

To prevent from fragmentation, **paging** was implemented. Paging is a memory management technique in which process address space is divided into blocks of the same size called *pages*. Moreover, main memory is also divided into physical memory blocks, with the same size, what is called *frames*.

Described above method called paging was used during this laboratory. There are several advantages from such approach:

1. reduces external fragmentation
2. simple to implement and efficient
3. swapping between pages and frames is easy due to the fact that they have equal sizes

SOLUTION

In order to configure mapping it was necessary to edit *memory.conf* file.

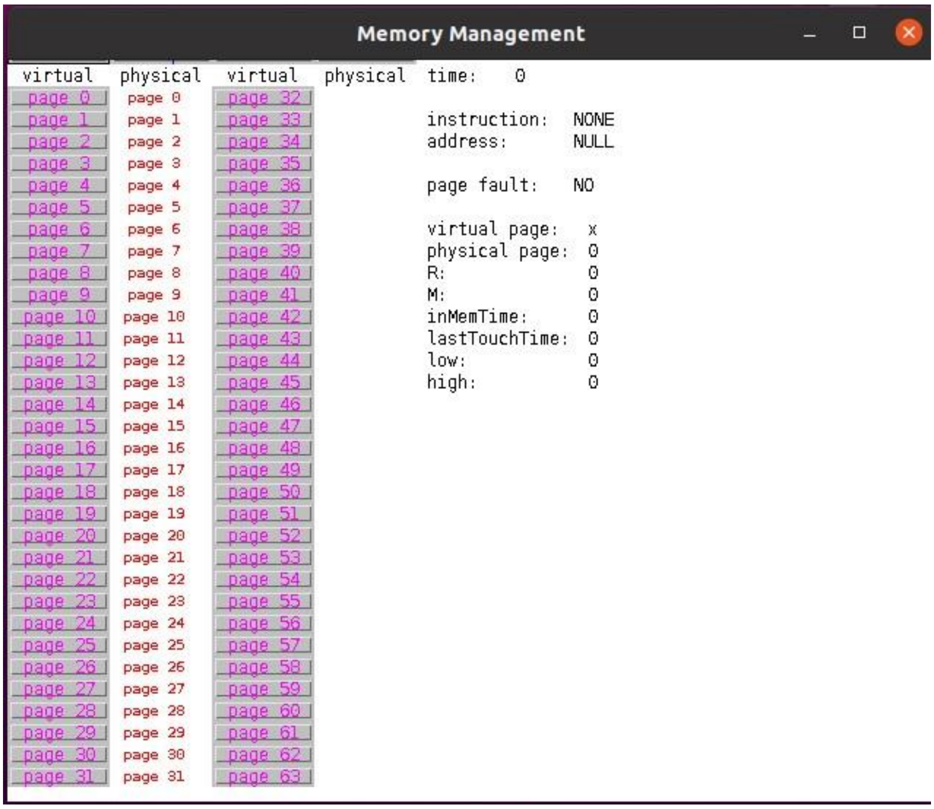
```
// memset  virt page #  physical page #  R (read from)  M (modified) inMemTime (ns) lastTouchTime (ns)
memset 0 0 0 0 0 0
memset 1 1 0 0 0 0
memset 2 2 0 0 0 0
memset 3 3 0 0 0 0
memset 4 4 0 0 0 0
memset 5 5 0 0 0 0
memset 6 6 0 0 0 0
memset 7 7 0 0 0 0

// 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
```

According to our task I map only 8 pages. It is done using *memset* command which assigns virtual pages to the physical pages respectively.

Secondly I configured *commands* file which specifies read operation for 64 virtual pages. In the *memory.conf* file, pagesize was set to 16384 addresses. Due to this fact I execute read command 64 times for addresses being multiples of 16384.

```
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
```

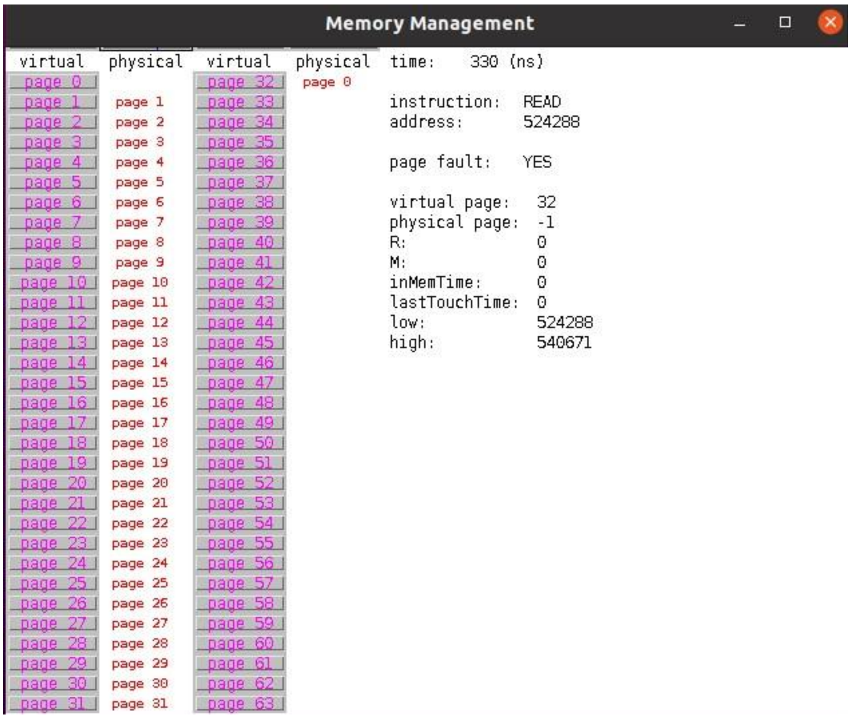
Below there is presented program running the simulation in its primary state. As it was stated in the *memory.conf* file, first 0-7 virtual pages correspond to first 0-7 physical pages. Rest of the pages are mapped by default.



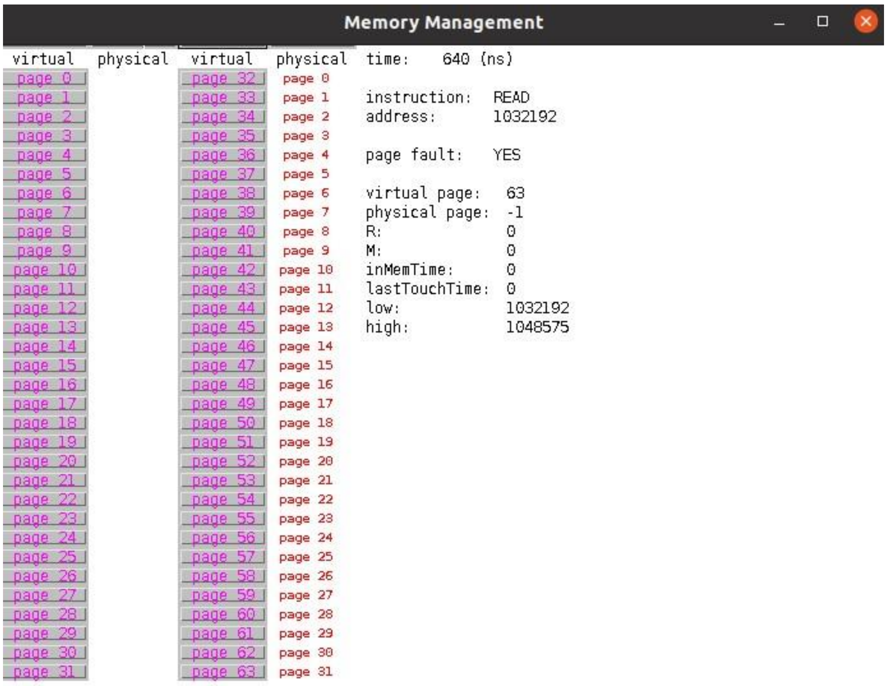
Next, if we start simulation by one step, we can notice that the first mapped page becomes the first replacement. This is caused due to the fact that the used page replacement algorithm was First-In First-Out (FIFO).

FIFO is the simplest page replacement algorithm. Operating System keeps track of all pages in the memory in a queue. In the front of the queue is the oldest page. When a page needs to be replaced, the first page from the queue is used.

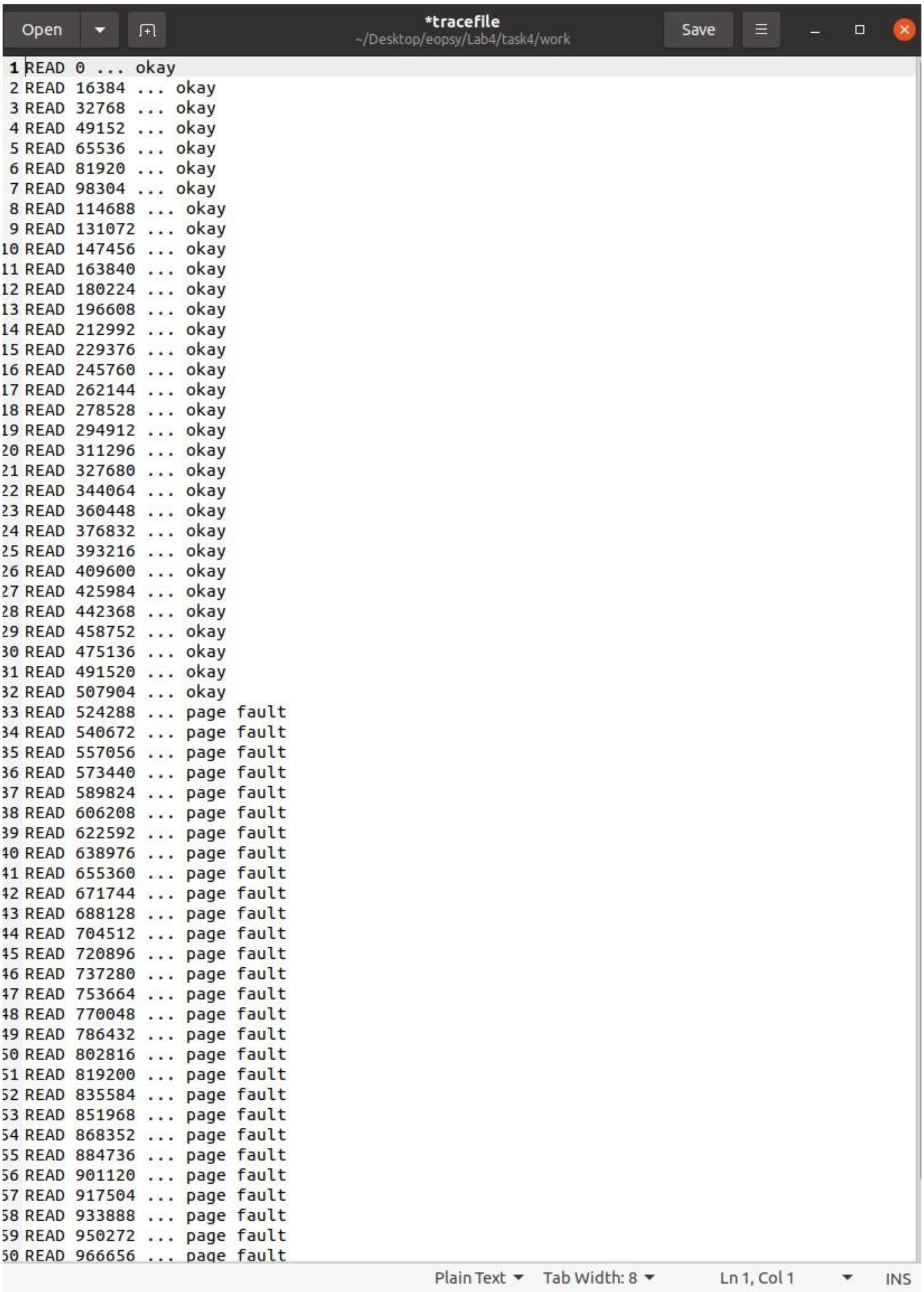
Due to the fact that there are 32 virtual pages mapped to physical pages and the rest virtual pages are not mapped to any physical pages, I expected faults while trying to access not mapped virtual pages. My suspicious have been confirmed and it can be seen on the picture below:



The first page fault occurs at the 32nd page. Page used as a replacement was the oldest page in the queue so the 0th page. The rest of the pages will be also replaced with the next page from the queue respectively. Below there is a result after going through all pages.



Moreover, the prove, that the first page fault occurs for the 32nd page, can be found in the *tracefile* with results. All pages below 32nd one caused also a fault:




```
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
```