Laboratory report

EOPSY LAB 4

Paweł Borsukiewicz

301106

Gitlab: https://gitlab-stud.elka.pw.edu.pl/pborsuki/eopsy-labs3-7.git

Table of contents:

Initial setup

```
//Map first 8 physical memory pages to first 8 virtual
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
```

Figure 1. Mapping

First of all, I have mapped any 8 pages of physical memory to the first 8 pages of virtual memory. In my case those “any pages were” first 8 pages, just for simplicity. Following four values defining if pages modification were left as 0 by default.

I have also left default size of page – 16384 (0x4000) – as there was no instruction to modify it.

```
pagesize 16384
```

Figure 2. Page size

So as to read one virtual from each of 64 virtual pages I have decided to start from address 0 (first address of virtual page #0) and then add page size to the address read.

```
//There is0x4000 adresses on each page

READ hex 0
READ hex 4000
READ hex 8000
READ hex c000
```

Figure 3. First four addresses read.

The last address read was 0xfc000 (first address of the virtual memory #63)

Results

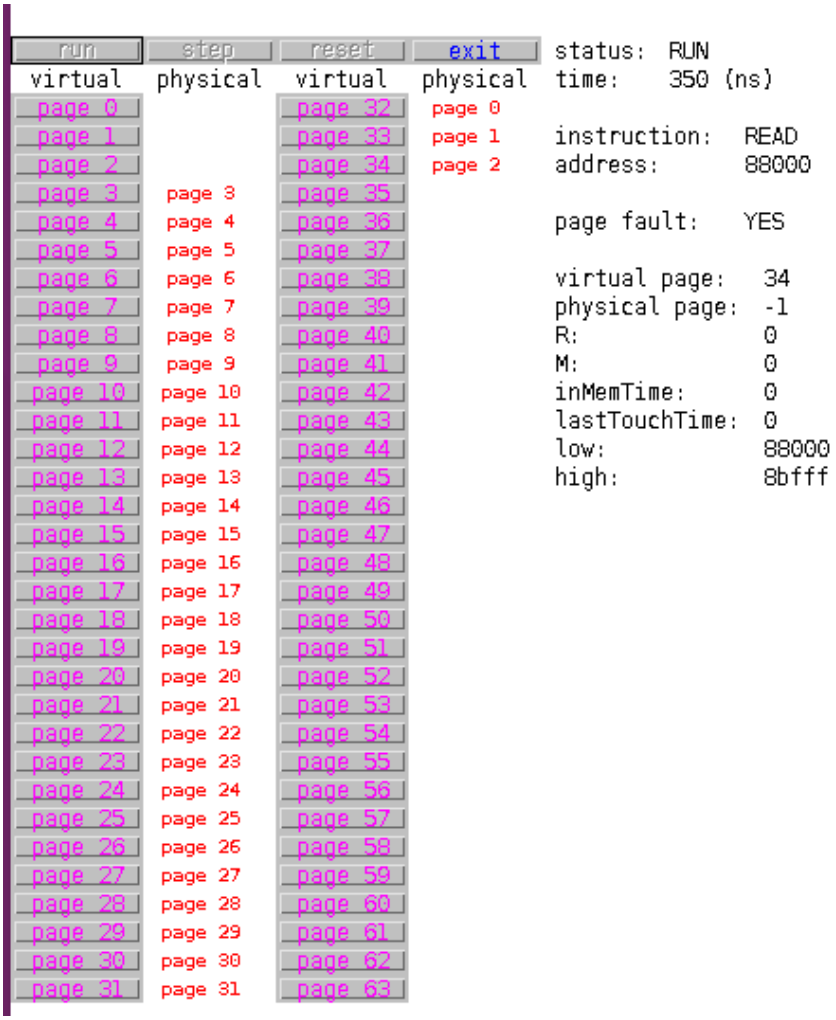

Figure 4. Simulation example

READ 0 ... okay
READ 4000 ... okay
READ 8000 ... okay
READ c000 ... okay
READ 10000 ... okay
READ 14000 ... okay
READ 18000 ... okay
READ 1c000 ... okay
READ 20000 ... okay
READ 24000 ... okay
READ 28000 ... okay
READ 2c000 ... okay
READ 30000 ... okay
READ 34000 ... okay
READ 38000 ... okay
READ 3c000 ... okay
READ 40000 ... okay
READ 44000 ... okay
READ 48000 ... okay
READ 4c000 ... okay
READ 50000 ... okay
READ 54000 ... okay
READ 58000 ... okay
READ 5c000 ... okay
READ 60000 ... okay
READ 64000 ... okay
READ 68000 ... okay
READ 6c000 ... okay
READ 70000 ... okay
READ 74000 ... okay
READ 78000 ... okay
READ 7c000 ... okay
READ 80000 ... page fault
READ 84000 ... page fault
READ 88000 ... page fault
READ 8c000 ... page fault
READ 90000 ... page fault
READ 94000 ... page fault
READ 98000 ... page fault
READ 9c000 ... page fault
READ a0000 ... page fault
READ a4000 ... page fault
READ a8000 ... page fault
READ ac000 ... page fault
READ b0000 ... page fault
READ b4000 ... page fault
READ b8000 ... page fault
READ bc000 ... page fault
READ c0000 ... page fault
READ c4000 ... page fault
READ c8000 ... page fault
READ cc000 ... page fault
READ d0000 ... page fault
READ d4000 ... page fault
READ d8000 ... page fault
READ dc000 ... page fault
READ e0000 ... page fault
READ e4000 ... page fault
READ e8000 ... page fault
READ ec000 ... page fault
READ f0000 ... page fault
READ f4000 ... page fault
READ f8000 ... page fault
READ fc000 ... page fault

Figure 5. Simulation tracefile

Analysis

To begin with, physical memory is a memory that refers to actual RAM of our device, while virtual memory is a way operating system can allow us to use more physical memory than available. It is based on paging and is slower than physical memory as it copies some data to/from hard disk, which is in general slower than reading from/writing to RAM.

I have noticed that even though I only link 8 physical pages to virtual pages, there are 32 physical pages total. Their initial configuration is as follows virtual page #n is connected with physical page #n, where n belongs to set of all integer numbers between 0 and 31. Virtual pages 32-64 are connected to page -1, which means that they are not connected.

While executing read commands, they are all successful until virtual memory #32 (0x80000) is to be accessed. On that moment we encounter first page fault. This exception is raised because the virtual page is not linked to any physical memory address (page). Similarly, all further virtual memory addresses fail. However, what is worth noticing is that after unsuccessful read instruction physical page is reassigned to the virtual memory page that could not be accessed (figure 4). Therefore, at the very end physical pages are assigned to virtual pages 32-63 (completely opposite to initial state).

This situation makes sense when one takes a look into contents of PageFault class. Fragment of comment, presented on figure 6, confirmed my suspicion that FIFO replacement algorithm was used. Within this algorithm operating systems remembers which pages were created recently and which one more in the past. All pages are queued according to their “age”. When page fault is encountered, oldest page slot is replaced with a new one – the one we tried to access. In our case the first physical page replaced and reallocated to virtual memory page #32 was page #0, which indicates that this page was the first one linked.

```
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.
```

Figure 6. Comments in PageFault class

Nonetheless, there are some other page replacement algorithms. For example LRU (least recently used), which replaces pages that has not been used for the longest time. Others such as MFU (most frequently used), LFU (least frequently used) decide on the number of times particular data was used and remove the one with the highest (MFU)/lowest(LFU) count.

Finally, trace file (figure 5) is generated and we can confirm that first 32 read instructions were successful, while last 32 were not.