EOPSY LAB4

Memory management

Michał Skarzyński 293054

TABLE OF CONTENT

Task 2
Package 2
Output explained 2
Input explained 2
Configuration File 2
Command File 3
initial configuration 4
General remarks 4
Configuration File 5
Command File 5
Initial mapping 6
Obtained results 6
Output file 8
RESULTS ANALYSYS 8

TASK

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.

PACKAGE

The memory management simulator illustrates page fault behavior in a paged virtual memory system. The program reads the initial state of the page table and a sequence of virtual memory instructions and writes a trace log indicating the effect of each instruction. It includes a graphical user interface so that students can observe page replacement algorithms at work. Students may be asked to implement a particular page replacement algorithm which the instructor can test by comparing the output from the student's algorithm to that produced by a working implementation.

OUTPUT EXPLAINED

The output file contains a log of the operations since the simulation started (or since the last reset). It lists the command that was attempted and what happened as a result. You can review this file after executing the simulation.

The output file contains one line per operation executed. The format of each line is:

```
command address ... status
```

where:

```
command is READ or WRITE,
address is a number corresponding to a virtual memory address, and
status is okay or page fault.
```

INPUT EXPLAINED

Configuration File

The configuration file ***memory.conf*** is used to specify the initial content of the virtual memory map (which pages of virtual memory are mapped to which pages in physical memory) and provide other configuration information, such as whether operation should be logged to a file.

The ***memset*** command is used to initialize each entry in the virtual page map. ***memset*** is followed by six integer values:

- The virtual page # to initialize
- The physical page # associated with this virtual page (-1 if no page assigned)
- If the page has been read from (R) (0=no, 1=yes)
- If the page has been modified (M) (0=no, 1=yes)
- The amount of time the page has been in memory (in ns)
- The last time the page has been modified (in ns)

```
memset virtual_id physical_id R M ns_in_memory ns_modify
```

The first two parameters define the mapping between the virtual page and a physical page, if any. The last four parameters are values that might be used by a page replacement algorithm.

Keyword	Values	Description
enable_logging	true/false	Whether logging of the operations should be enabled. If logging is enabled, then the program writes a one-line message for each READ or WRITE operation. By default, no logging is enabled. See also the log_file option.
log_file	trace-file-name	The name of the file to which log messages should be written. If no filename is given, then log messages are written to stdout. This option has no effect if enable_logging is false or not specified.
pagesize	n power p	The size of the page in bytes as a power of two. This can be given as a decimal number which is a power of two (1, 2, 4, 8, etc.) or as a power of two using the power keyword. The maximum page size is 67108864 or power 26. The default page size is power 26.
addressradix	n	The radix in which numerical values are displayed. The default radix is 2 (binary). You may prefer radix 8 (octal), 10 (decimal), or 16 (hexadecimal).
numpages	Int from <2, 64>	Sets the number of pages (physical and virtual)

Command File

There are two operations one can carry out on pages in memory: READ and WRITE.

The format for each command is

```
operation address
```

where operation is READ or WRITE, and address is the numeric virtual memory address, optionally preceded by one of the radix keywords bin, oct, or hex. If no radix is supplied, the number is assumed to be decimal.

INITIAL CONFIGURATION

General remarks

The first thing that catches the eye after starting the program is the fact that it is inconsistent with the documentation.

The description of the parameters is simply not true, the biggest problem is the ***numpages*** parameter which according to the documentation would determine the number of virtual and physical pages, unfortunately it does not work as described, number of virtual pages is actually equal to this parameter, but unfortunately the number of physical pages is the value of this parameter divided in half. This is a special problem because it is **NOT POSSIBLE** to declare 8 physical and 64 virtual pages. Possible configurations are: 8 physical pages, 16 virtual, or 32 physical 64 virtual. Completing the task as stated is therefore impossible to do.

Assuming a case in which we have 8 physical and 16 virtual pages of memory, it is obviously not possible to execute the READ / WRITE command from the page above page id 16, which contradicts checking each address on all 64 pages of virtual memory.

According to the configuration file, the memset command takes the virtual page number as the first argument and the physical page number as the second argument - this is inconsistent with what is displayed in the user interface, the mapping in columns is inverted. Interestingly, after displaying detailed information about the virtual page, it is assigned to the correct physical page.

The documentation also claims that "If a virtual page is not specified by any memset command, it is assumed that the page is not mapped." this is simply not true because if we do not declare the memset command for a given page of physical memory, it is automatically mapped to the first unassigned virtual page. Setting this parameter to -1 does not help either, which should be a solution to this problem as we can read in another part of the documentation: "The physical page # associated with this virtual page (-1 if no page assigned)".

Unfortunately, it is not possible to make the configuration in the program as expected in the task.

Therefore in my task, I assume that:

- There are 8 pages of physical memory
- There are 16 pages of virtual memory
- The authoritative mapping is assigned in the configuration file and displayed in the detailed information after clicking on the page, not shown in the columns
- The check will be changed to check the address from each of the 16 pages of virtual memory instead of, as specified in task, 64

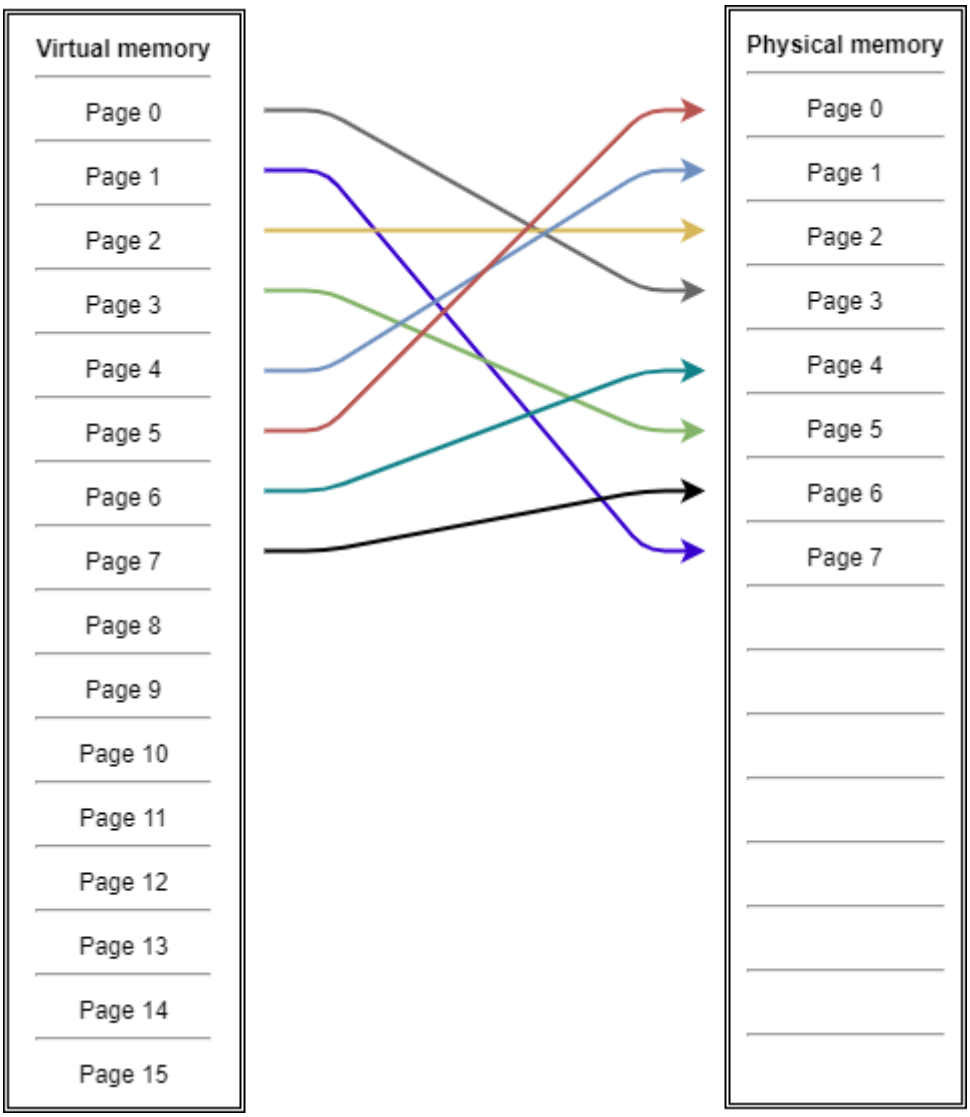
Configuration File

```
memset 0 3 0 0 0 0
memset 1 7 0 0 0 0
memset 2 2 0 0 0 0
memset 3 5 0 0 0 0
memset 4 1 0 0 0 0
memset 5 0 0 0 0 0
memset 6 4 0 0 0 0
memset 7 6 0 0 0 0

enable_logging true

log_file tracefile

pagesize 16384

addressradix 16

numpages 16
```

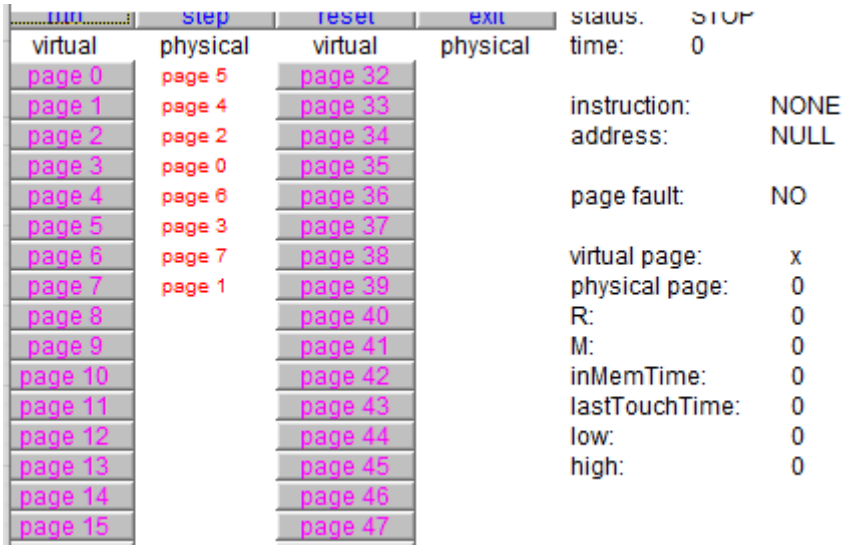
Command File

```
READ hex 0001
READ hex 4001
READ hex 8010
READ hex C000
READ hex 10001
READ hex 14002
READ hex 18004
READ hex 1c003
READ hex 20002
READ hex 24002
READ hex 28002
READ hex 2c002
READ hex 30002
READ hex 34002
READ hex 38002
READ hex 3c000
```

Initial mapping

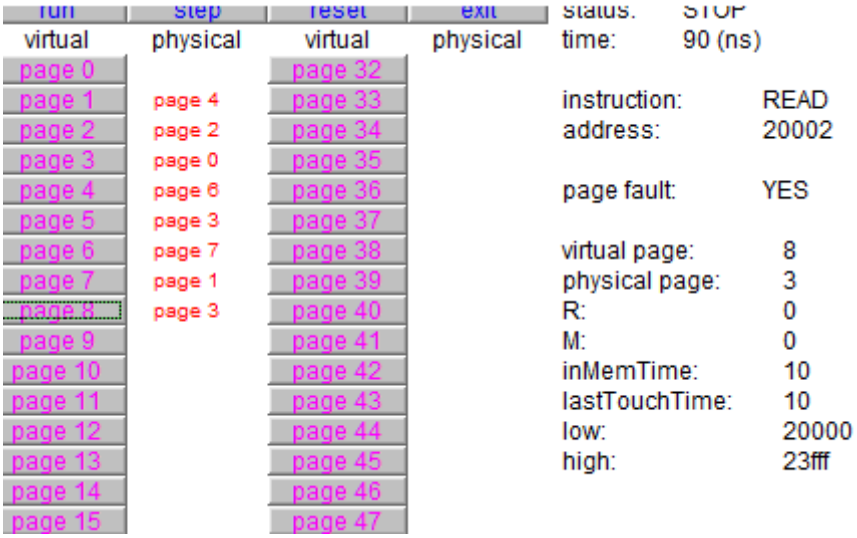


OBTAINED RESULTS



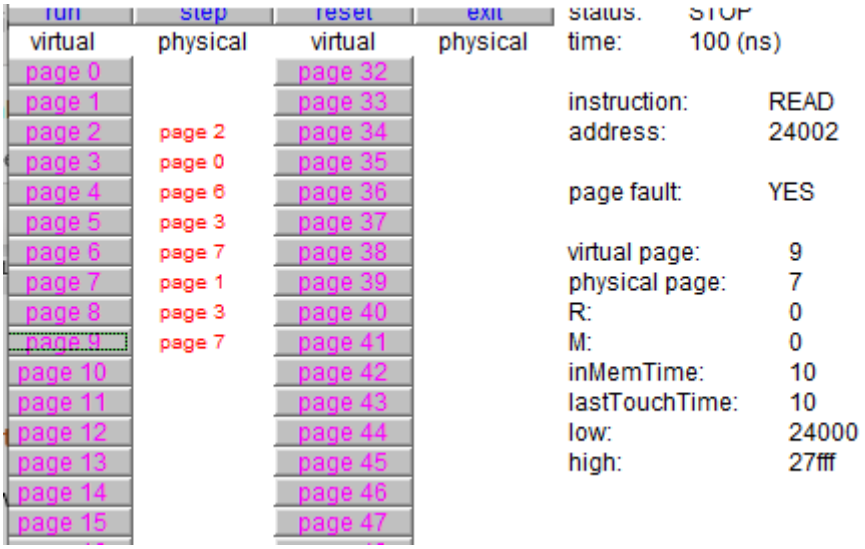
Please note that the mapping displayed in the columns is not the actual mapping. It is reversed.

Up to step 8 nothing will change – since we only access pages which are already in memory

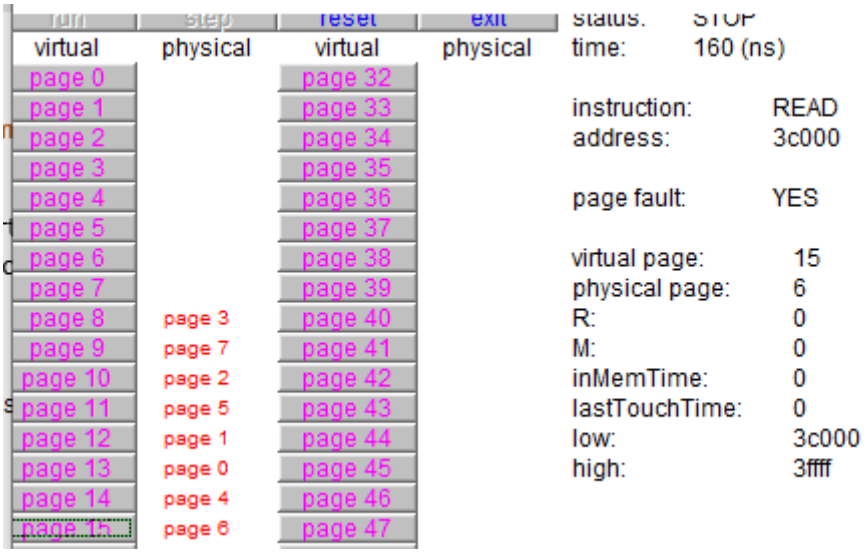


Please note that the mapping which occurred during execution is the displayed properly. But all pages from initial mapping are reversed.

First page fault appears when we want to access memory on virtual page 8, then last page (page 0 which was mapped to physical page 3) is removed, and this space is assigned to page 8.



This process will now continue for every other page



Up to page 15, where execution stops.

Output file

```
READ 1 ... okay
READ 4001 ... okay
READ 8010 ... okay
READ c000 ... okay
READ 10001 ... okay
READ 14002 ... okay
READ 18004 ... okay
READ 1c003 ... okay
READ 20002 ... page fault
READ 24002 ... page fault
READ 28002 ... page fault
READ 2c002 ... page fault
READ 30002 ... page fault
READ 34002 ... page fault
READ 38002 ... page fault
READ 3c000 ... page fault
```

RESULTS ANALYSYS

First page fault appears after try to access memory in virtual page 8, which was to be expected since this page is not mapped to any physical page.

Then algorithm decides to remove mapping for virtual page 0 and use freed space, namely page 3 in physical memory, to assign it into page 7.

This will then continue for every next page.

If program wants to access address from virtual memory page which is not assigned to any physical memory page then:

- Delete mapping for the oldest assigned virtual memory page.
- Use this space in physical memory for assigning new virtual memory page.

We can therefore conclude that this is FIFO algorithm – all pages are queued using their time which they have spent in memory, the oldest page is the one in front of the queue – one to be removed on the next page fault.