EOPSY Task 4: Memory Management

Piotr Kochański 289372

I declare that this piece of work which is the basis of recognition of achieving learning outcomes in the EOPSY course was completed on my own.

Piotr Kochański 289372

Virtual memory is an abstraction over the actual memory - physical memory. The virtual address space is mapped to the physical address space by software (kernel) and hardware (MMU). The benefits of virtual memory are that applications are freed from having to manage shared memory space and can allocate more memory that is physically available on the computer, memory for shared objects can be reused, and also increased security due to memory isolation.

A page fault is a type of exception generated by computer hardware when the page that is being accessed is not currently mapped to a virtual page of the process - not loaded in the physical memory. This happens because virtual address space is much larger than the physical one.

When a page fault occurs a page replacement algorithm is run to determine which page will be replaced. Different approaches to page replacement algorithms are used based on the workflow required. The goal is to reduce the amount of page faults and thus increase performance.

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.

```
// 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 16

// 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
```

With the memset command we are assigning physical pages to virtual pages. In this example to the virtual pages 0-7 (first column) are being assigned physical pages 0-7 (second column).

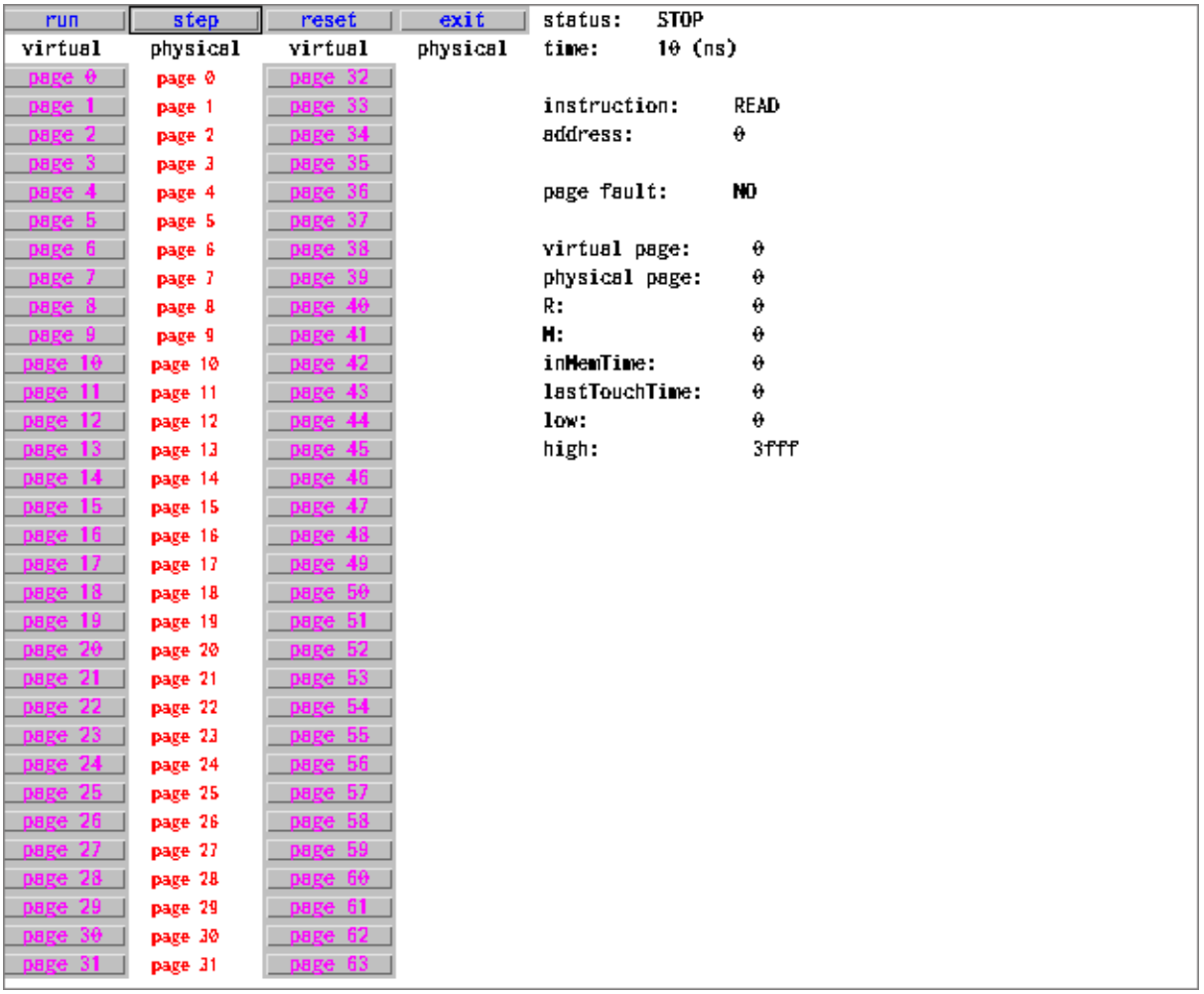
```
// Enter READ/WRITE commands into this file
// READ <OPTIONAL number type: bin/hex/oct> <virtual memory address or random>
// WRITE <OPTIONAL number type: bin/hex/oct> <virtual memory address or random>
READ hex 00000
READ hex 04000
READ hex 08000
READ hex 0c000
READ hex 10000
READ hex 14000
READ hex 18000
READ hex 1c000
READ hex 20000
READ hex 24000
READ hex 28000
READ hex 2c000
READ hex 30000
READ hex 34000
READ hex 38000
READ hex 3c000
```

…

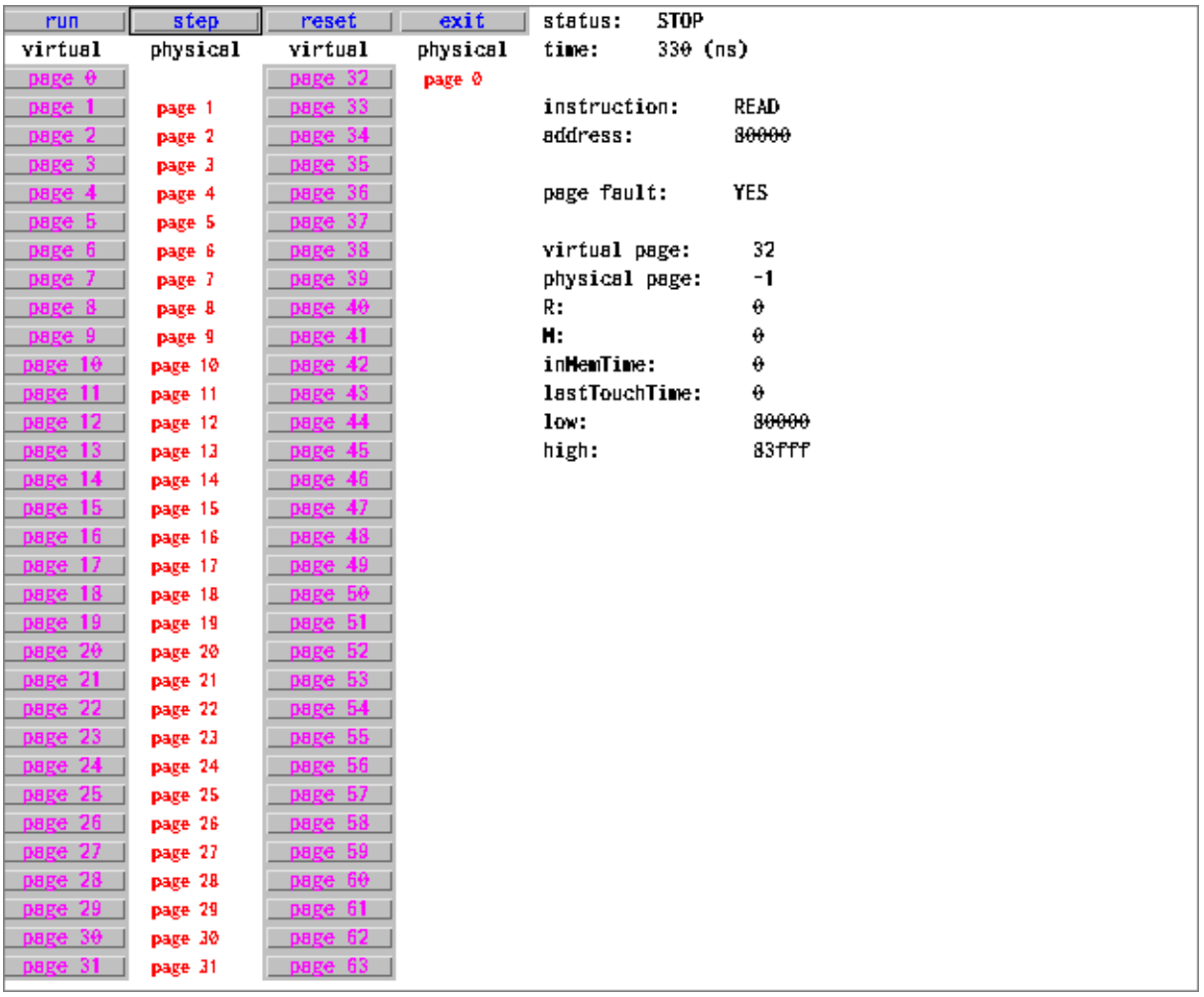
```
READ hex d0000
READ hex d4000
READ hex d8000
READ hex dc000
READ hex e0000
READ hex e4000
READ hex e8000
READ hex ec000
READ hex f0000
READ hex f4000
READ hex f8000
READ hex fc000
```

0x0400 is 16384 in decimal - the page size - each read is from the first address of the page.

Step through the simulator one operation at a time and see if you can predict which virtual memory addresses cause page faults.



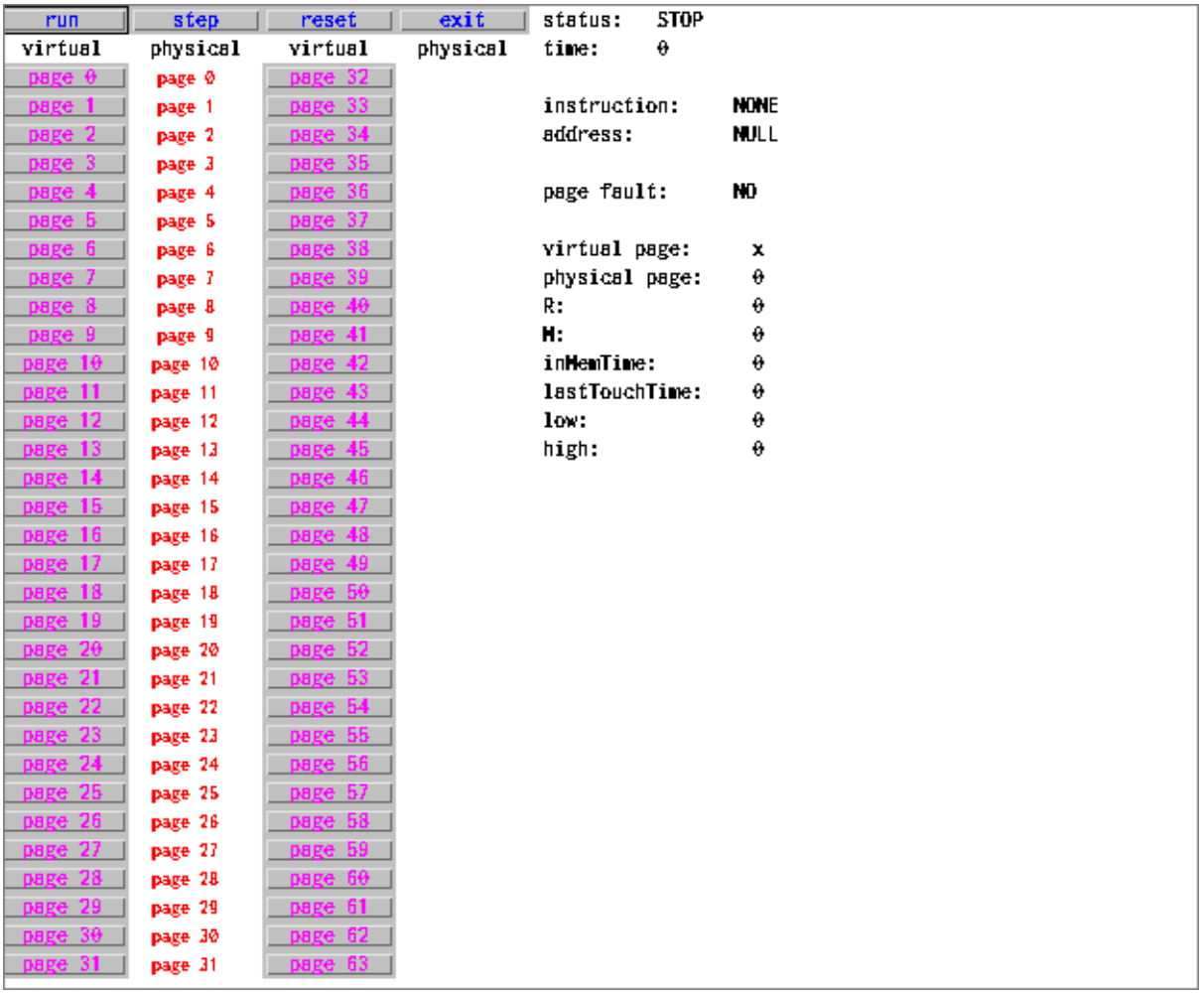
By default even if only 8 pages of virtual memory are assigned the rest of the 32 virtual pages are assigned to the first 32 pages of physical memory.



And as one would suspect first page fault is encountered at address 80000 - that is page 32. Page 32 and all the following pages were not assigned to the virtual page. Accessing them causes page fault and the first assigned virtual page is getting assigned to a new physical page (FIFO).

```
// 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
memset 8 -1 0 0 0 0
memset 9 -1 0 0 0 0
memset 10 -1 0 0 0 0
memset 11 -1 0 0 0 0
memset 12 -1 0 0 0 0
memset 13 -1 0 0 0 0
memset 14 -1 0 0 0 0
memset 15 -1 0 0 0 0
memset 16 -1 0 0 0 0
memset 17 -1 0 0 0 0
memset 18 -1 0 0 0 0
memset 19 -1 0 0 0 0
memset 20 -1 0 0 0 0
memset 21 -1 0 0 0 0
memset 22 -1 0 0 0 0
memset 23 -1 0 0 0 0
memset 24 -1 0 0 0 0
memset 25 -1 0 0 0 0
memset 26 -1 0 0 0 0
memset 27 -1 0 0 0 0
memset 28 -1 0 0 0 0
memset 29 -1 0 0 0 0
memset 30 -1 0 0 0 0
memset 31 -1 0 0 0 0
```

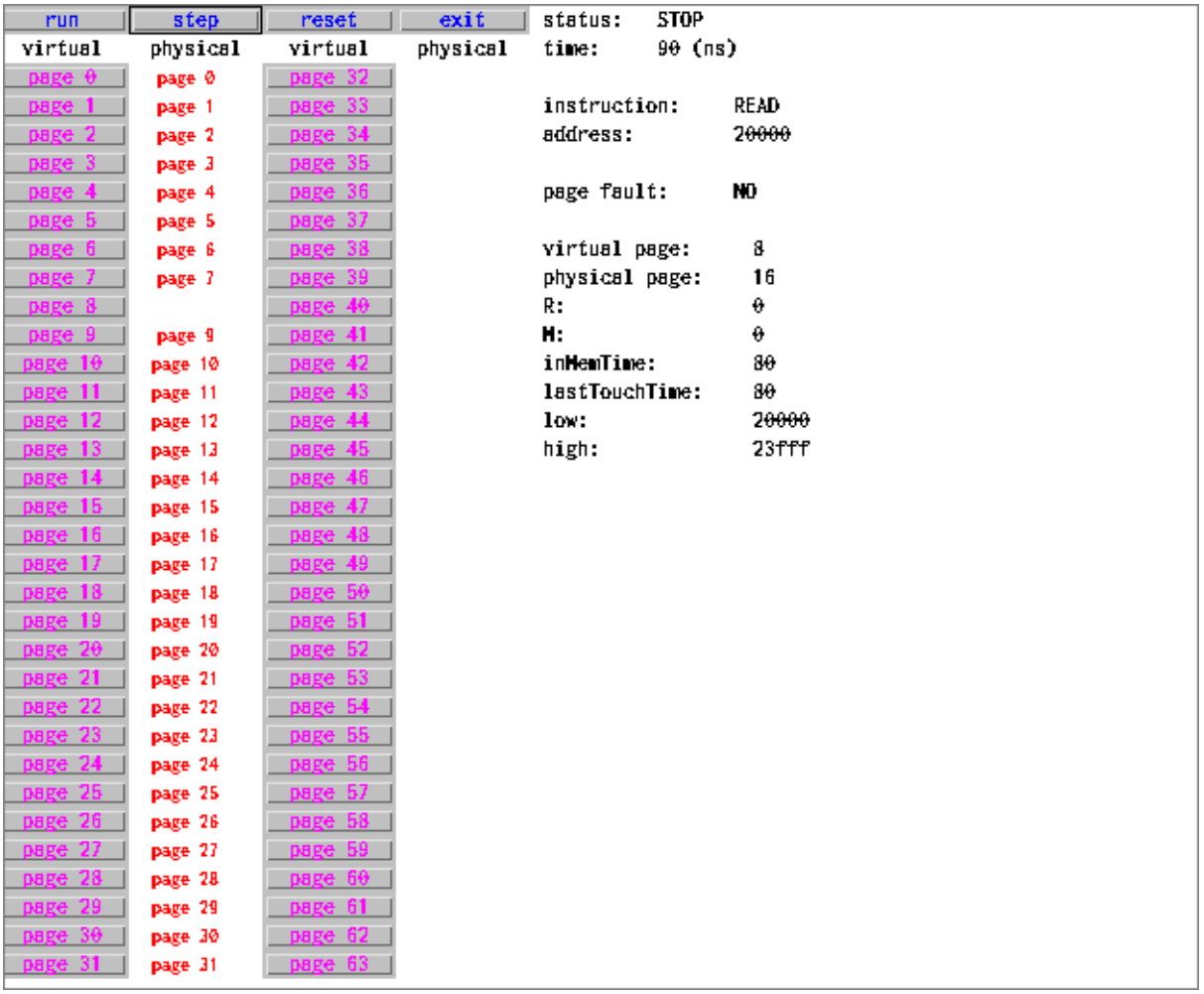
Unfortunately even if we set -1 (no page assigned) to the other virtual pages they still get apparently assigned.

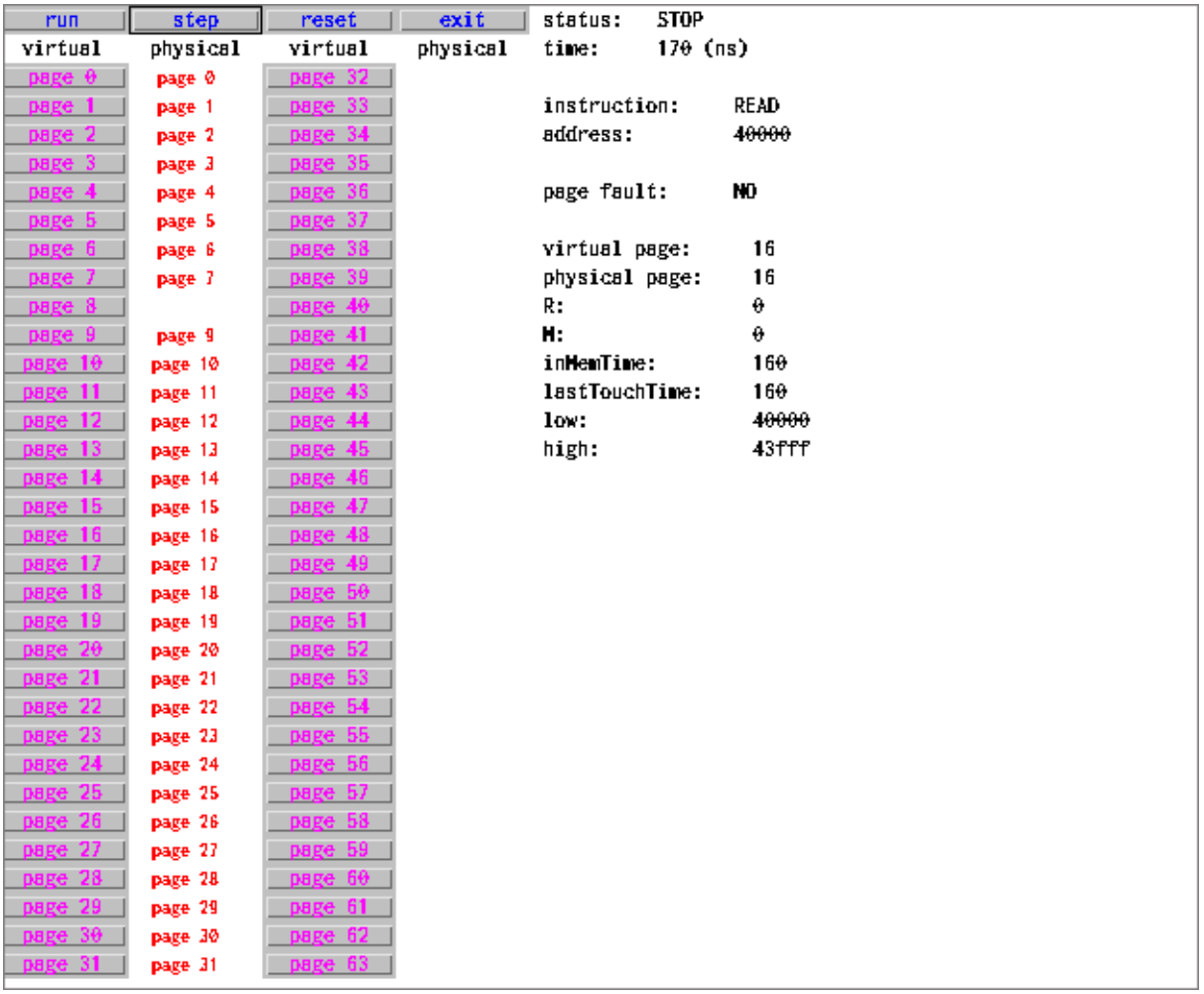


Also it is not possible to assign pages from outside of the 0-31 range (even though 32-63 are valid pages), and if there is a gap in page assignments, and pages in the gap are not explicitly assigned:

```
memset 7 7 0 0 0 0
memset 8 16 0 0 0 0
```

it results in visually missing page:


run
step
reset
exit
status: STOP
virtual
physical
virtual
physical
time: 90 (ns)
page 0
page 0
page 32
page 1
page 1
page 33
instruction: READ
page 2
page 2
page 34
address: 20000
page 3
page 3
page 35
page 4
page 4
page 36
page fault: NO
page 5
page 5
page 37
page 6
page 6
page 38
virtual page: 8
page 7
page 7
page 39
physical page: 16
page 8
page 40
R: 0
page 9
page 9
page 41
M: 0
page 10
page 10
page 42
inMemTime: 80
page 11
page 11
page 43
lastTouchTime: 80
page 12
page 12
page 44
low: 20000
page 13
page 13
page 45
high: 23fff
page 14
page 14
page 46
page 15
page 15
page 47
page 16
page 16
page 48
page 17
page 17
page 49
page 18
page 18
page 50
page 19
page 19
page 51
page 20
page 20
page 52
page 21
page 21
page 53
page 22
page 22
page 54
page 23
page 23
page 55
page 24
page 24
page 56
page 25
page 25
page 57
page 26
page 26
page 58
page 27
page 27
page 59
page 28
page 28
page 60
page 29
page 29
page 61
page 30
page 30
page 62
page 31
page 31
page 63



but actually both page 8 and page 16 are assigned to 16 (this must be an error in the software).

What page replacement algorithm is being used?

```
public static void replacePage ( Vector mem , int virtPageNum , int replacePageNum , 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;
}
```

Page replacement algorithm used is first in first out page replacement - the oldest mapped page is mapped to the new page. The algorithm can be found in PageFault.java. The algorithm scans the list of virtual pages looking for pages that are not assigned a physical page. Once it finds a page without a physical page assigned it stores that in the firstPage variable. The virtual pages not assigned physical pages are checked for the time in memory - the longest time in memory and the page with it is stored, thus the oldest page is found. If there is no oldest page the first unmapped page (firstPage) is used.