Memory Management Simulation Interactive Lab Answer Key

Experiments

1. Keep memory.conf as is. Modify the commands file by entering the following sequence of commands:

```
// 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 bin 110
READ bin 111
WRITE hex CB33
WRITE hex FB12
WRITE hex B4A2B
READ bin 100000100100000
READ bin 110000010000110
WRITE bin 110011100000000
WRITE random
```

Now, try running the simulator (type java MemoryManagement commands memory.conf from a command prompt). Press the reset button and then the run button. Take a look at the log file. *The log file is shown below (Figure 1).* Are there any page faults? *Yes.*

If so, where do these occur, and why? *The fifth line (WRITE b4a2b) is virtual page 45, which is not mapped to a physical page. You can tell this by taking a look at the configuration file, where you can see that the last virtual page mapped is 31. The last operation is a random write, which may or may not cause a page fault.*

```
READ 6 ... okay
READ 7 ... okay
WRITE cb33 ... okay
WRITE fb12 ... okay
WRITE b4a2b ... page fault
READ 4120 ... okay
READ 6086 ... okay
WRITE 6700 ... okay
```

Figure 1

2. Modify the commands file again by entering the following sequence of commands:

READ bin 100
READ bin 010
READ bin 111
WRITE hex cc12
WRITE hex bc35
WRITE random
READ bin 111110100000
WRITE 6001
WRITE hex 7563e

Now, try running the simulator (type java MemoryManagement commands memory.conf from a command prompt). Press the reset button and then the run button. Take a look at the log file. Are there any page faults? *Yes.*
If so, where do these occur, and why?
Your results may vary slightly. As long as the write random causes a page fault, then there will be one page fault there. Figure 2 shows the simulator after the page fault. Notice how physical page 0 has now been mapped to virtual page 35. When we try to read virtual page 0 in the next command, notice how there is no longer an associated physical page (see Figure 3). Your log file should look similar to Figure 4. If there was no page fault caused by the random write, then the next read (READ bin 111110100000) should not have a page fault.

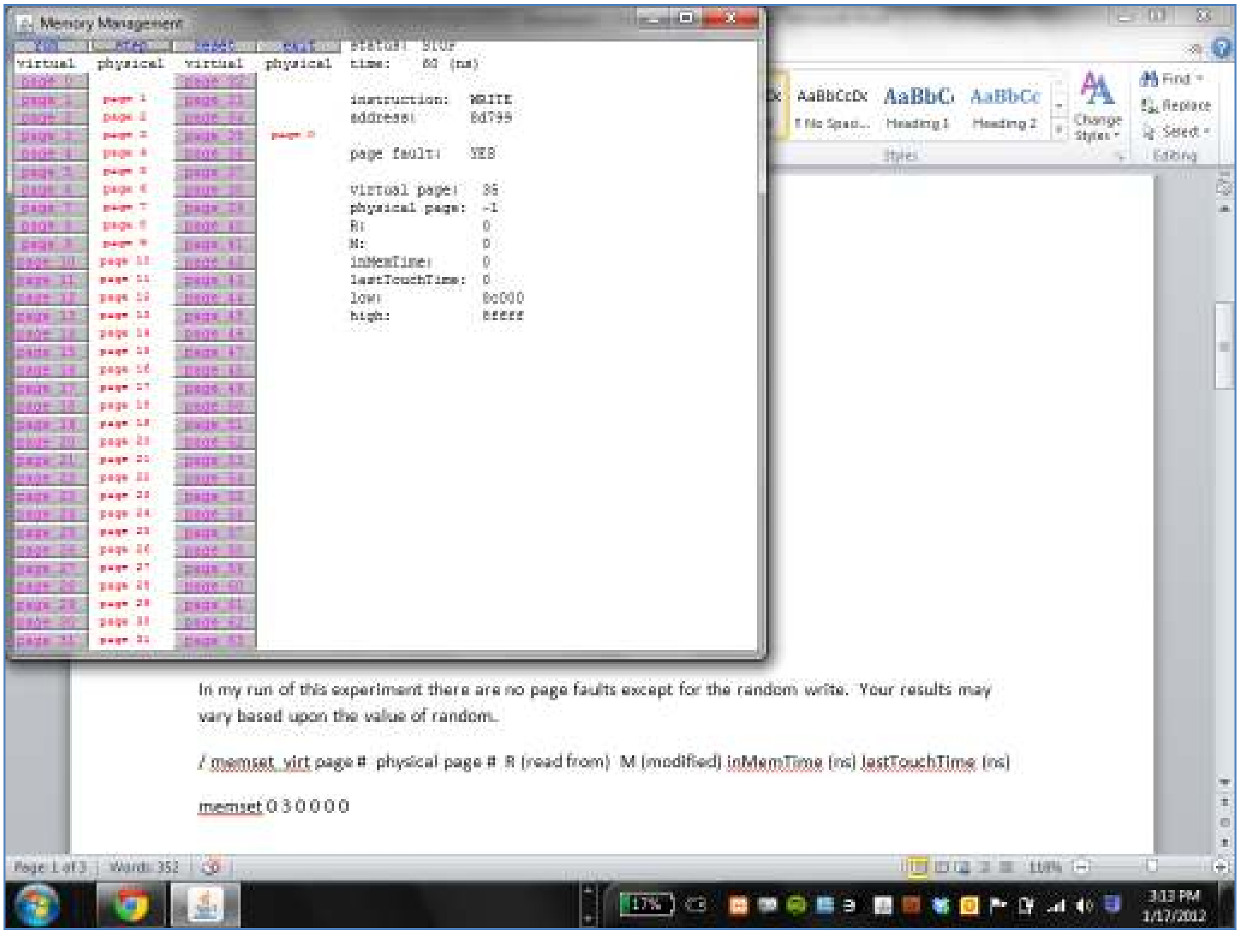
Memory Management
virtual physical virtual physical
time: 60 (ns)
instruction: WRITE
page fault: YES
virtual page: 35
physical page: -1
R: 0
M: 0
inMemTime: 0
lastTouchTime: 0
In my run of this experiment there are no page faults except for the random write. Your results may vary based upon the value of random.
/ memset virt page # physical page # R (read from) M (modified) inMemTime (ns) lastTouchTime (ns)
memset 0 3 0 0 0 0

Figure 2

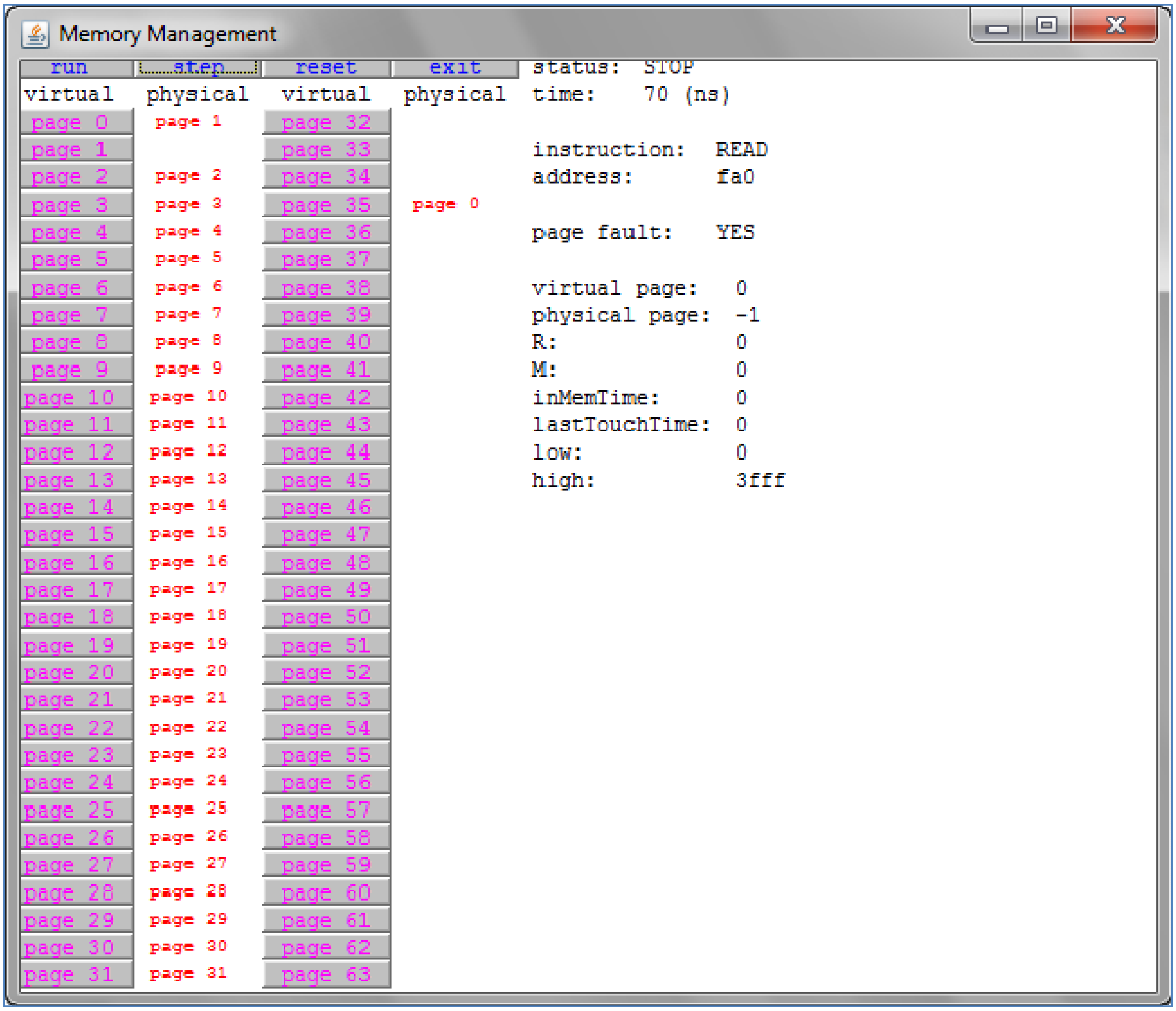

Figure 3

```
READ 4 ... okay
READ 2 ... okay
READ 7 ... okay
WRITE cc12 ... okay
WRITE bc35 ... okay
WRITE 8d799 ... page fault
READ fa0 ... page fault
WRITE 1771 ... okay
WRITE 7563e ... okay
```

Figure 4

3. Consider a virtual memory system with a page size of 1024. There are eight virtual pages and four physical frames. The page table is shown below:

Virtual Page Number	Page Frame Number
0	3
1	1
2	--
3	--
4	2
5	--
6	0
7	--

Keep a copy of the original memory.config file. Modify the memory.config file to reflect the page table above. Compare your file to the answer key. *Please see Figure 5 below.*

```
// memset  virt page #  physical page #  R (read from)  M (modified) inMemTime
(ns) lastTouchTime (ns)
memset 0 3 0 0 0 0
memset 1 1 0 0 0 0
memset 2 -1 0 0 0 0
memset 3 -1 0 0 0 0
memset 4 2 0 0 0 0
memset 5 -1 0 0 0 0
memset 6 0 0 0 0 0
memset 7 -1 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 1024

// 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 9
```

Figure 5

Modify the commands file to test the following operations:
READ 750

WRITE 1301
READ 2560
READ 4018
WRITE 4495
READ 5180
READ 6437
READ 7263

Which of these virtual addresses cause page fault? Why?
2560 needs to access virtual page 2, which does not have a physical page. Figure 6 shows the simulator after the page fault. Now, virtual page 2 maps to physical page 3. Virtual page 0 maps to no physical page.

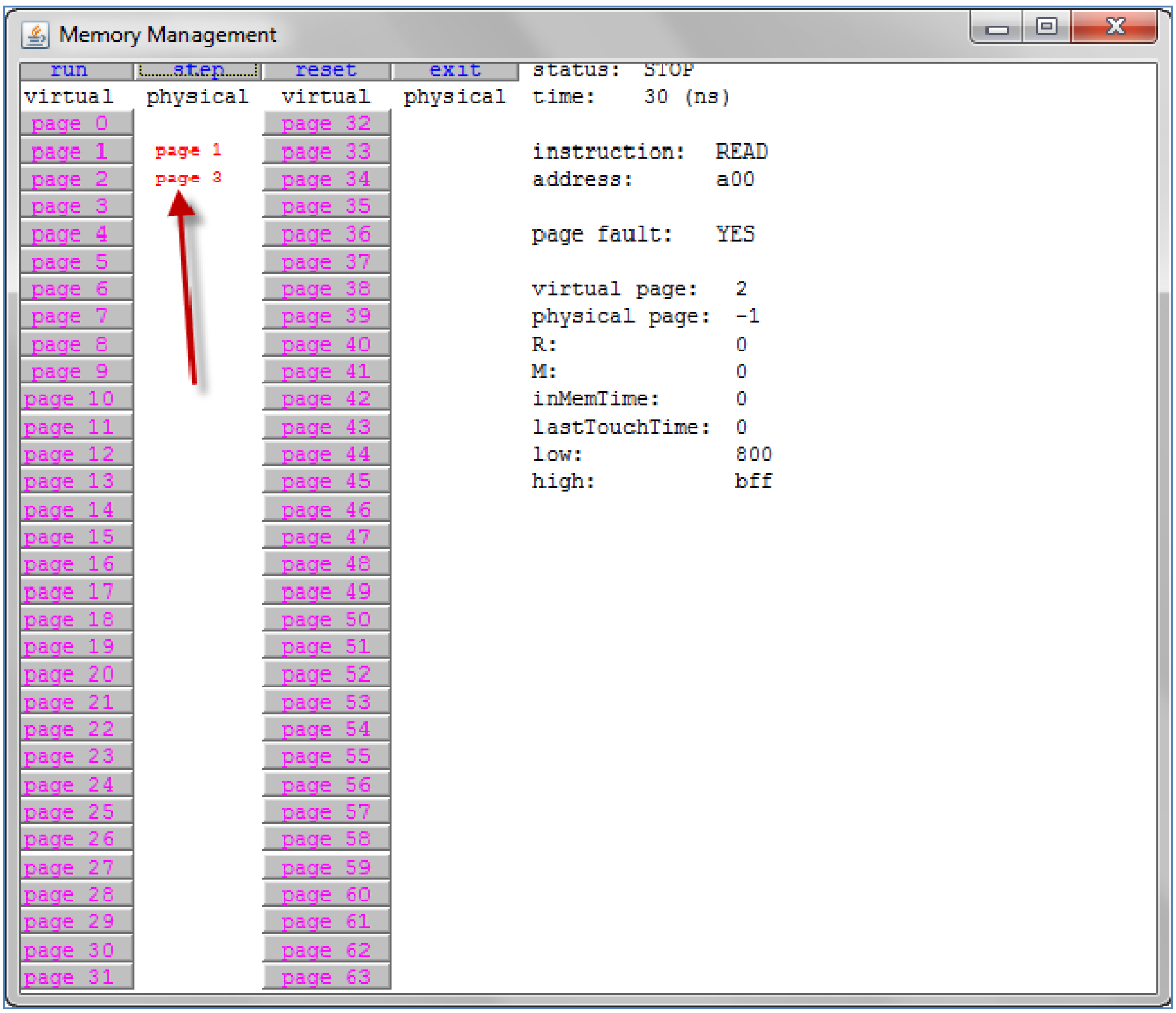

Figure 6

4018 needs to access virtual page 3, which has no physical page assigned to it. This causes a page fault. Figure 7 shows that virtual page 3 no has physical page 1 assigned to it.

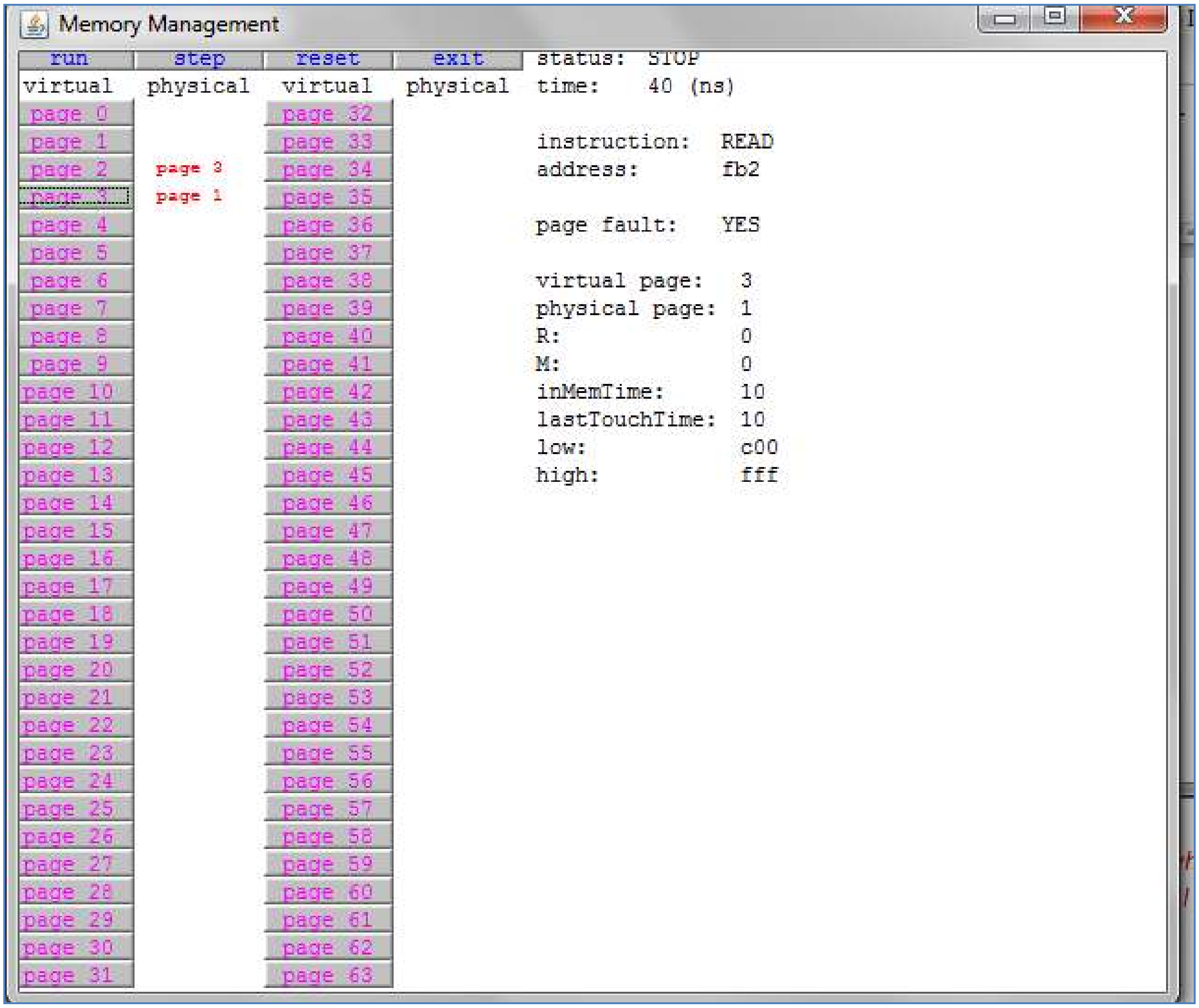

Figure 7

5180 is on virtual page 5, and there is no physical page assigned to it. After the page fault, physical page 2 is assigned to virtual page 5.

7263 is on virtual page 7, which has no physical page assigned. After the page fault, physical page 0 is assigned to this virtual page.

When running the simulation, the thing to keep in mind is that a page fault will change the original mapping. You need to pay attention to the simulator screen to keep track of this.

4. Modify a copy of the original memory.config file to map any 8 pages of physical memory to the first 8 pages of virtual memory. Modify a copy of the original commands file to read from one virtual memory address on each of the 64 virtual pages. Run the simulator in single step mode. Which virtual memory addresses caused page faults? Compare your answers to the answer key.
Your results will vary; however, what I am showing here is a representative sample. Figure 8 shows my memory.config file:

```
// memset  virt page #  physical page #  R (read from)  M (modified) inMemTime
(ns) lastTouchTime (ns)
memset 0 12 0 0 0 0
memset 1 1 0 0 0 0
memset 2 23 0 0 0 0
memset 3 11 0 0 0 0
memset 4 15 0 0 0 0
memset 5 5 0 0 0 0
memset 6 3 0 0 0 0
memset 7 9 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
```

Figure 8

I used the following commands file:

READ 11386
READ 22383
READ 37141
READ 59601
READ 78117
READ 85765
READ 99924
READ 119460
READ 133556
READ 154951
READ 174278
READ 185627
READ 212108
READ 213915
READ 235100
READ 259602
READ 266951
READ 285726
READ 295471
READ 313990
READ 334896
READ 358839
READ 371307
READ 379050
READ 407997
READ 419199
READ 436136
READ 455435
READ 464743
READ 484808
READ 495559
READ 520154
READ 527247
READ 544486
READ 571445
READ 574648

READ 601959
READ 608242
READ 634464
READ 650334
READ 665303
READ 680123
READ 700084
READ 718045
READ 736765
READ 752113
READ 764461
READ 772474
READ 797201
READ 811811
READ 823332
READ 851304
READ 865084
READ 873704
READ 898206
READ 915878
READ 927862
READ 936529
READ 951949
READ 978808
READ 990300
READ 1008584
READ 1022333

When the simulator runs, the mapping will be done as specified. The simulator also maps out the remaining virtual pages up to 31. Any memory request to a virtual page over 31 will cause a page fault.

Question

1. Based on what you have seen with the experiments, what page replacement algorithm is being used by the MOSS memory management simulator?

   First-in First-out, which services each request sequentially.