EOPSY LAB – Task 4 Report

Qingqing Yan

Task:

1. Get the necessary package from: https://www.ia.pw.edu.pl/~tkruk/edu/eopsy/lab/task4.tgz
2. Follow instructions from file: README.tjk (file is available within the package – in folder work):

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.

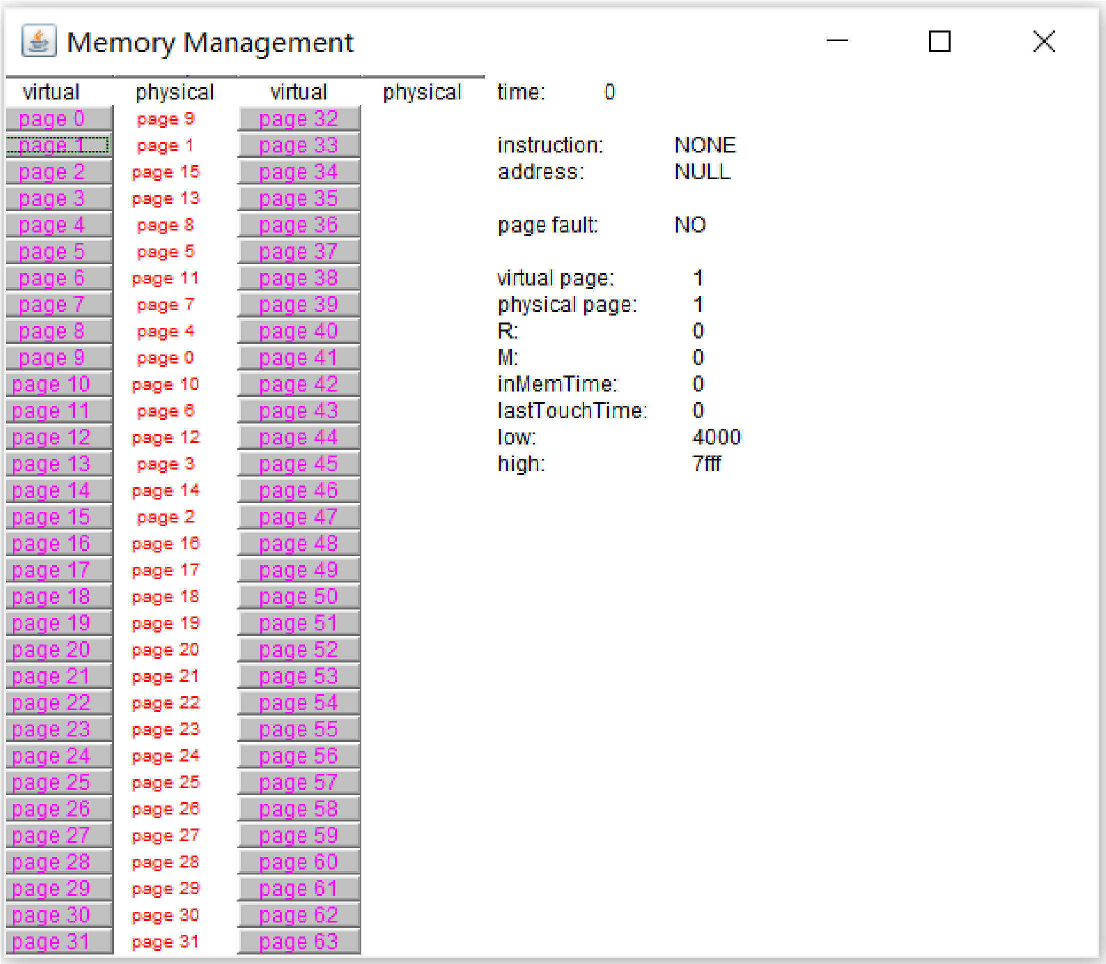
1. Create a command file that maps any 8 pages of physical memory to the first 8 pages of virtual memory, and then reads from one virtual memory address on each of the 64 virtual pages.

```
//memory.conf:
memset 0 9 0 0 0 0
memset 1 1 0 0 0 0
memset 2 15 0 0 0 0
memset 3 13 0 0 0 0
memset 4 8 0 0 0 0
memset 5 5 0 0 0 0
memset 6 11 0 0 0 0
memset 7 7 0 0 0 0
memset 9 0 0 0 0 0
memset 11 6 0 0 0 0
memset 13 3 0 0 0 0
memset 15 2 0 0 0 0
memset 8 4 0 0 0 0
```

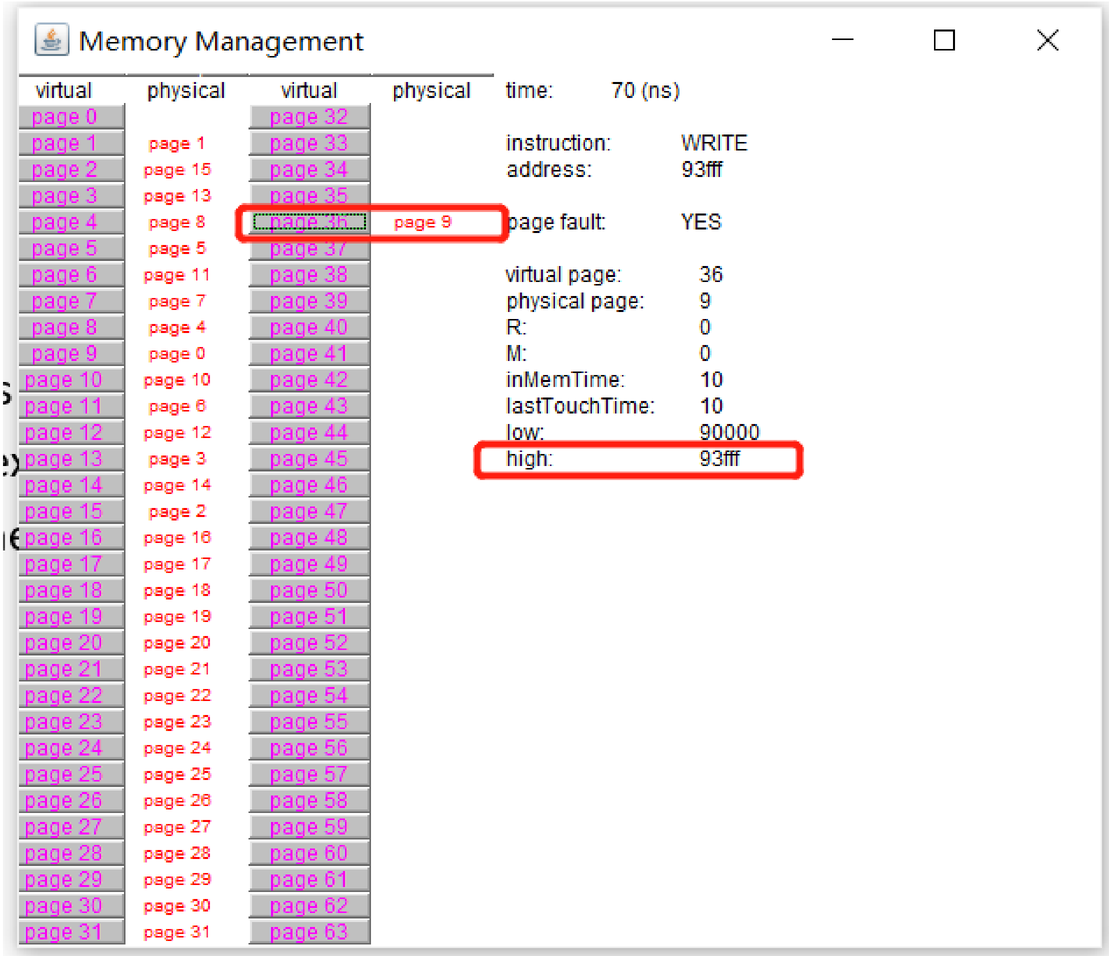
```
//commands:
// Enter READ/WRITE commands into this file
// READ <OPTIONAL number type: bin/hex/oct> <virtual memory address or random>
// WRITE <OPTIONAL number type: bin/hex/oct> <virtual memory address or random>
READ bin 100
READ 19
WRITE hex CC32
READ bin 10000000000000000
READ bin 10000000000000000
WRITE bin 11000000000000001
WRITE hex 93fff
```

2.Befor running program :

We can see the virtual address page 0 is mapped to physical memory page 9.



3.After running program



And the tracefile is:

READ 4 ... okay

READ 13 ... okay

WRITE cc32 ... okay

READ 10000 ... okay

READ 10000 ... okay

WRITE 18001 ... okay

WRITE 93fff ... page fault

93fff is the high address of the virtual address page 36.

We can see a memory page page 36 that is mapped into the virtual address space(93fff) has been loaded in physical memory page 9.

2. reads from 64 virtual memory addresses on each of the 64 virtual pages.

//memory.conf:

memset 0 9 0 0 0 0

memset 1 1 0 0 0 0

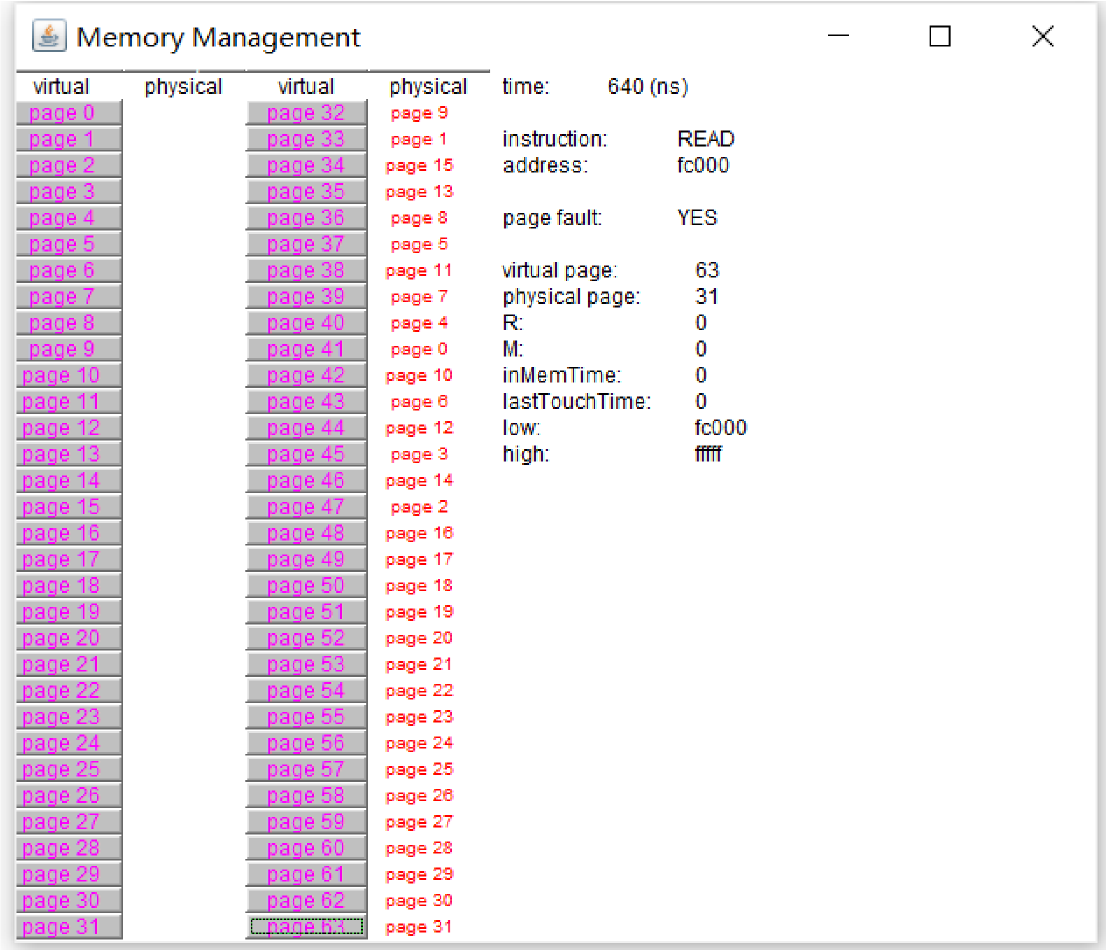
memset 2 15 0 0 0 0

memset 3 13 0 0 0 0

memset 4 8 0 0 0 0

memset 5 5 0 0 0 0
memset 6 11 0 0 0 0
memset 7 7 0 0 0 0
memset 9 0 0 0 0 0
memset 11 6 0 0 0 0
memset 13 3 0 0 0 0
memset 15 2 0 0 0 0
memset 8 4 0 0 0 0
//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 2
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
Execution result:



the tracefile is:

READ 2 ... 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

Concusion:
In this task, the configure simulator maps 32 physical memory pages to 32 pages of virtual memory address on each of the 64 virtual pages. If we don't map virtual memory address to physical memory ourselves, then the configure simulator will automatically map 32 pages of virtual memory address to 32 pages of physical memory one by one, for example: virtual memory page 0 is mapped to the physical memory page 0, virtual memory page 20 is mapped to the physical memory page 20, and so on. For the virtual memory pages from 32 to 63 page, there is no enough physical memory to map the virtual memory pages from 32 to 63 page.
From the execution result, we can observe that page fault happened:

1. WRITE 93fff ... page fault

93fff is virtual memory 36th page, In the memory management page, we can observe that the virtual page 36th(address 93fff) is mapped to the physical memory page 9 which is the first physical memory page mapped to the virtual memory page 0.

2. In the tracefile, after the virtual page 31 ,all page faults happened. when our program accesses the memory pages that are mapped into the virtual address space(from 32 to 61 page), but not loaded in physical memory, page faults happened. We can see that the virtual memory page 0 to 31 are replaced by virtual memory address 32 to 63 which are loaded into the physical memory page 0 to 31. Namely the newly needed pages(32th page to 63th page) will replace the existing pages. The newly needed page will be loaded into the oldest physical memory page. The new pages will sequentially replace the oldest pages in the physical memory in the order.

3.So, according to the execution result, in this task, First In First Out (FIFO) is used.
Page Replacement Algorithms: First In First Out (FIFO):
This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.