

CI283 Operating Systems

Memory Management Simulator

Purpose

In this exercise, you will use a Memory Management Simulator. This guide explains how to use the simulator and describes the display and the various input files used and the output files produced by the simulator.

The memory management simulator illustrates page fault behaviour 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 you can observe page replacement algorithms at work.

Download **memory.zip** from Studentcentral and unzip it.


Running the Simulator

The program reads a command file, optionally reads a configuration file, displays a GUI window which allows you to execute the command file, and optionally writes a trace file.

To run the program, first compile all Java files and then enter the following command line.

```
$ java MemoryManagement commands memory.conf
```

The program will display a window allowing you to run the simulator. You will notice a row of command buttons across the top, two columns of "page" buttons on the left, and an informational display on the right.


Memory Management

run	step	reset	exit	status: STOP
virtual	physical	virtual	physical	time: 0
page 0	page 0	page 32		
page 1	page 1	page 33		instruction: NONE
page 2	page 2	page 34		address: NULL
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: x
page 7	page 7	page 39		physical page: 0
page 8	page 8	page 40		R: 0
page 9	page 9	page 41		M: 0
page 10	page 10	page 42		inMemTime: 0
page 11	page 11	page 43		lastTouchTime: 0
page 12	page 12	page 44		low: 0
page 13	page 13	page 45		high: 0
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		

Typically, you will use the `step` button to execute a command from the input file, examine information about any pages by clicking on a `page` button, and when you're done, quit the simulation using the `exit` button.

The buttons:

Button	Description
run	runs the simulation to completion. Note that the simulation pauses and updates the screen between each step.

step	runs a single setup of the simulation and updates the display.
reset	initializes the simulator and starts from the beginning of the command file.
exit	exits the simulation.
page <i>n</i>	display information about this virtual page in the display area at the right.

The informational display:

Field	Description
status:	RUN, STEP, or STOP. This indicates whether the current run or step is completed.
time:	number of "ns" since the start of the simulation.
instruction:	READ or WRITE. The operation last performed.
address:	the virtual memory address of the operation last performed.
page fault:	whether the last operation caused a page fault to occur.
virtual page:	the number of the virtual page being displayed in the fields below. This is the last virtual page accessed by the simulator, or the last page <i>n</i> button pressed.
physical page:	the physical page for this virtual page, if any. -1 indicates that no physical page is associated with this virtual page.
R:	whether this page has been read. (1=yes, 0=no)
M:	whether this page has been modified. (1=yes, 0=no)
inMemTime:	number of ns ago the physical page was allocated to this virtual page.
lastTouchTime:	number of ns ago the physical page was last modified.
low:	low virtual memory address of the virtual page.
high:	high virtual memory address of the virtual page.

The Command File

The command file for the simulator specifies a sequence of memory instructions to be performed. Each instruction is either a memory READ or WRITE operation, and includes a virtual memory address to be read or written. Depending on whether the virtual page for the address is present in physical memory, the operation will succeed, or, if not, a page fault will occur.

Operations on Virtual Memory

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

The format for each command is

operation address

or

operation random

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. The keyword `random` will generate a random virtual memory address (for those who want to experiment quickly) rather than having to type an address.

For example, the sequence

```
READ bin 01010101
WRITE bin 10101010
READ random
WRITE random
```

causes the virtual memory manager to:

1. read from virtual memory address 85
2. write to virtual memory address 170
3. read from some random virtual memory address
4. write to some random virtual memory address

Sample Command File

The "commands" input file looks like this:

```
// Enter READ/WRITE commands into this file
// READ
// WRITE
READ bin 100
READ 19
WRITE hex CC32
READ bin 1000000000000000
READ bin 1000000000000000
WRITE bin 1100000000000001
WRITE random
```

The Configuration File

The configuration file `memory.conf` is used to specify the 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.

Setting Up the Virtual Memory Map

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

1. The virtual page # to initialize

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

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.

For example,

```
memset 34 23 0 0 0 0
```

specifies that virtual page 34 maps to physical page 23, and that the page has not been read or modified.

Note:

- Each physical page should be mapped to exactly one virtual page.
- The number of virtual pages is fixed at 64 (0..63).
- The number of physical pages cannot exceed 64 (0..63).
- If a virtual page is not specified by any `memset` command, it is assumed that the page is not mapped.

Other Configuration File Options

There are a number of other options which can be specified in the configuration file. These are summarized in the table below.

Keyword	Values	Description
<code>enable_logging</code>	<code>true</code> <code>false</code>	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 <code>log_file</code> option.
<code>log_file</code>	<i>trace-file-name</i>	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 <code>enable_logging</code> is false or not specified.
<code>pagesize</code>	<i>n</i> power <i>p</i>	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.
<code>addressradix</code>	<i>n</i>	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).

Sample Configuration File

The "memory.conf" configuration file looks like this:

```
// 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 8 0 0 0 0
memset 9 9 0 0 0 0
memset 10 10 0 0 0 0
memset 11 11 0 0 0 0
memset 12 12 0 0 0 0
memset 13 13 0 0 0 0
memset 14 14 0 0 0 0
memset 15 15 0 0 0 0
memset 16 16 0 0 0 0
memset 17 17 0 0 0 0
memset 18 18 0 0 0 0
memset 19 19 0 0 0 0
memset 20 20 0 0 0 0
memset 21 21 0 0 0 0
memset 22 22 0 0 0 0
memset 23 23 0 0 0 0
memset 24 24 0 0 0 0
memset 25 25 0 0 0 0
memset 26 26 0 0 0 0
memset 27 27 0 0 0 0
memset 28 28 0 0 0 0
memset 29 29 0 0 0 0
memset 30 30 0 0 0 0
memset 31 31 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
// Where  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 or <'power' num (base 2)>
pagesize 16384

// addressradix sets the radix in which numerical
values are displayed
// 2 is the default value
// addressradix
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
numpages 64
```

The Output File

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.

Sample Output

The output "tracefile" looks something like this:

```
READ 4 ... okay
READ 13 ... okay
WRITE 3acc32 ... okay
READ 10000000 ... okay
READ 10000000 ... okay
WRITE c0001000 ... page fault
WRITE 2aeea2ef ... okay
```

Suggested Exercises

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. 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?