1 Memory Management Simulator

Please find the memory management source files from the moodle. This 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.

To make things easier for you, we have implemented the FIFO page replacement algorithm already. Please go through the implementation carefully so that you can work out how to write your own page replacement algorithms. Please go through the instructions carefully and complete the assignment.

2 Running the simulator

- Compile the java code using the following command.

```
$ javac *.java
```

- The program reads a command file, configuration file, and writes a trace file.
- You can run the program by running the following command.

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

- 'commands' refers to the file where we state the command sequence we need to run on the system.
- 'Memory.conf' file has the initial configuration of the system.(i.e: 'memory_FIFO.conf')

2.1 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.

2.1.1 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 the operation is READ or WRITE, and the 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

2.2 The 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 the operation should be logged to a file.

2.2.1 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 number to initialize
2. The physical page number 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 default number of virtual pages is 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.
- '`memset`' commands must be defined at the end of the configuration file.

2.2.2 Other Configuration File Options

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

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. 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).
physicalMemSize	*n*	The size of the physical memory as a measurement of the number of pages.
replacementAlgorithm	*FIFO |LRU |Clock_policy*	The page replacement algorithm to use in the simulator.

2.3 The Output File

The output file contains a log of the operations since the simulation started. 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.
- `status` is okay or page fault.

Example:

```
READ 10000000 ... okay
READ 10000000 ... okay
WRITE c0001000 ... page fault
```

3 Assignment

3.1 Task 1

- Read and understand the simulator and the implementation of the FIFO algorithm.
- Run the program with the '`commands`' file and the '`memory_FIFO.conf`' file.
- Identify how the FIFO algorithm works.

3.2 Task 2

- Implement Least recently used(LRU) page replacement algorithm in the '`PageFault.java`' and call it within the '`replacePage()`' method.
- Use the '`tracefile_LRU`' as a reference for what your output should look like when you run the program with the '`commands`' file and the '`memory_LRU.conf`' file.

3.3 Task 3

- Implement the Clock-policy page replacement algorithm in the '`PageFault.java`' and call it within the '`replacePage()`' method.
- Use the '`tracefile_clock`' as a reference for what your output should look like when you run the program with the '`commands`' file and the '`memory_clock.conf`' file.

4 Submission

- You do not have to worry about the input type of the addresses while implementing page replacement algorithms since all the addresses are converted and saved as decimal numbers by the kernel.
- Submit the '`PageFault.java`' file to the submission link in the moodle before the deadline.
- Please keep the code clean and add comments. There will be marks for the code quality and comments.

Your submission will be tested against inputs that we have designed.

- **Do NOT** change any source file other than the '`PageFault.java`'.
- **Do NOT** change the function interfaces of any functions in the '`PageFault.java`'. Any change will result in your code failing the tests.
- If you need more static variables for your implementation you can define them **without changing** other data structures inside the '`PageFault.java`'.
- **Do NOT** output anything other than what has been asked for. If you have added any outputs for your convenience, you should remove/comment them **before submission**.