MOSS Memory Management Simulator Installation on Unix/Linux/Solaris/HP-UX Systems

Purpose

This document provides instructions for the installation of the MOSS Memory Management Simulator on Unix operating systems. This procedure should be the same or similar on Unix, Linux, Solaris, HP-UX and other Unix-compatible systems. The MOSS software is designed for use with Andrew S. Tanenbaum, Modern Operating Systems, 2nd Edition (Prentice Hall, 2001). The Memory Management Simulator was written by Alex Reeder (*alexr@e-sa.org*). This installation guide was written by Ray Ontko (*rayo@ontko.com*).

Requirements

The following software components are required to install and use the MOSS Memory Management Simulator.

- X-windows environment for running Java Application Window Toolkit (AWT) programs
- Java Development Kit (JDK) 1.0 or greater
- Text program editor (e.g., notepad)

Pre-Installation

Before installation, you should verify:

- that you have a working java runtime environment,
- that you have a working java development environment, and
- that the working directory is in the classpath for the runtime environment.

If you're using a standard command-line java compiler, the following instructions will help determine if your environment is configured correctly.

1. Verify that you have java installed and configured in your environment.

   ```
   $ java -version
   ```

   You should see a message like this with possibly a different version number.

   ```
   java version "1.1.8"
   ```

   If you get a message like:

   ```
   java: Command not found.
   ```

   Then java may not be installed on your system, or may not be configured for your use.

   If you think that Java may already be installed on your system but may not be in your "path", you can find it by

   ```
   $ find /usr -name java -print
   ```

   On my system, for example, the following is returned.

   ```
   /usr/lib/netscape/477/communicator/java
   /usr/lib/netscape/477/netscape/java
   /usr/lib/jdk1.1/bin/java
   /usr/lib/jdk1.1/bin/ia32/green_threads/java
   /usr/share/java
   /usr/bin/java
   /usr/src/kernel-source-2.2.17/include/config/binfmt/java
   ```

   On my system, I also searched for "javac" and found that it exists in /usr/bin/java. I'll use this jdk for my installation.

   If Java isn't available on your system, you should check with your instructor or system administrator. If you administer your own system, then you should be able to find a copy of Java for your operating system.

   If you find that java is installed but not configured for your use, then perhaps you need to add it to your path. Consult your instructor or system administrator if you need help adding this to your path.

2. Verify that the java compiler is installed and configured in your environment.

   ```
   $ javac
   ```

   If you're using a standard java command-line compiler, you should see a message similar to this.

   ```
   use: javac [-g][-O][-debug][-depend][-nowarn][-verbose][-classpath path][-nowrite][-deprecation][-d dir][-J] file.java...
   ```

   If you get a message like:

   ```
   javac: Command not found.
   ```

   then the java compiler may not be installed on your system, or may not be configured for your use. Consult your instructor or system administrator.

3. Verify that that the current directory is in your classpath.

   ```
   $ echo $CLASSPATH
   ```

   You should see a list of directories separated by colons (":") or possibly just a blank line. If you don't see the directory "." (a single period, which stands for the current directory), then you should add it to the claspath.

   Determine which shell you're using:

   ```
   $ echo $SHELL
   ```

   If you're using sh, ksh, or bash:

   ```
   $ CLASSPATH=.:$CLASSPATH
   $ export CLASSPATH
   ```

   If you're using csh, or tcsh:

   ```
   % set CLASSPATH=.:$CLASSPATH
   ```

If you have a working java runtime environment, a working java compiler, and the current directory is in your path, then you're ready to proceed with the installation.

Installation

Installation of the software can be accomplished with these simple steps:

1. Create a directory in which you wish to install the simulator (e.g., "moss/memory").

   ```
   $ cd
   $ mkdir moss
   $ cd moss
   $ mkdir memory
   $ cd memory
   ```

2. Download the compressed tar archive (memory.tgz) into the directory. The latest release for this file can always be found at http://www.ontko.com/moss/memory/memory.tgz.

3. Expand the compressed tar archive.

   ```
   $ tar -xzf memory.tgz
   ```

   or

   ```
   $ gunzip memory.tgz
   $ tar xf memory.tar
   ```

Files

The directory should now contain the following files:

Files	*Description*
memory.tgz	Compressed tar archive which contains all the other files.
Common.java ControlPanel.java Instruction.java Kernel.java MemoryManagement.java PageFault.java Page.java Virtual2Physical.java	Java source files (*.java)
Common.class ControlPanel.class Instruction.class Kernel.class MemoryManagement.class PageFault.class Page.class Virtual2Physical.class	Compiled Java class files (*.class)
commands	Sample input command file
memory.conf	Sample configuration file
install_unix.html install_windows.html user_guide.html user_guide_1.gif	Documentation and associated images
javadoc	Directory containing documentation on java classes (generated by the javadoc utility).
copying.txt	Gnu General Public License: Terms and Conditions for Copying, Distribution, and Modification

Compilation

The distribution includes compiled class files as well as the source java files. You should not need to recompile unless you decide to change the code. If you wish to compile the code, the following commands should work if you're using a Java compiler that accepts the normal "javac" command line.

```
$ javac -nowarn *.java
```

The -nowarn flag supresses warning messges, of which there may be several. For backward compatability we use only those features of Java which have been present from the beginning, some of which are deprecated and are usually reported by the compiler with warning messages.

MOSS Memory Management Simulator User Guide

Purpose

This document is a user guide for the MOSS Memory Management Simulator. It explains how to use the simulator and describes the display and the various input files used by and output files produced by the simulator. The MOSS software is designed for use with Andrew S. Tanenbaum, Modern Operating Systems, 2nd Edition (Prentice Hall, 2001). The Memory Management Simulator was written by Alex Reeder (*alexr@e-sa.org*). This user guide was written by Ray Ontko (*rayo@ontko.com*).

Introduction

The memory management simulator illustrates page fault behavior 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 students can observe page replacement algorithms at work. Students may be asked to implement a particular page replacement algorithm which the instructor can test by comparing the output from the student's algorithm to that produced by a working implementation.

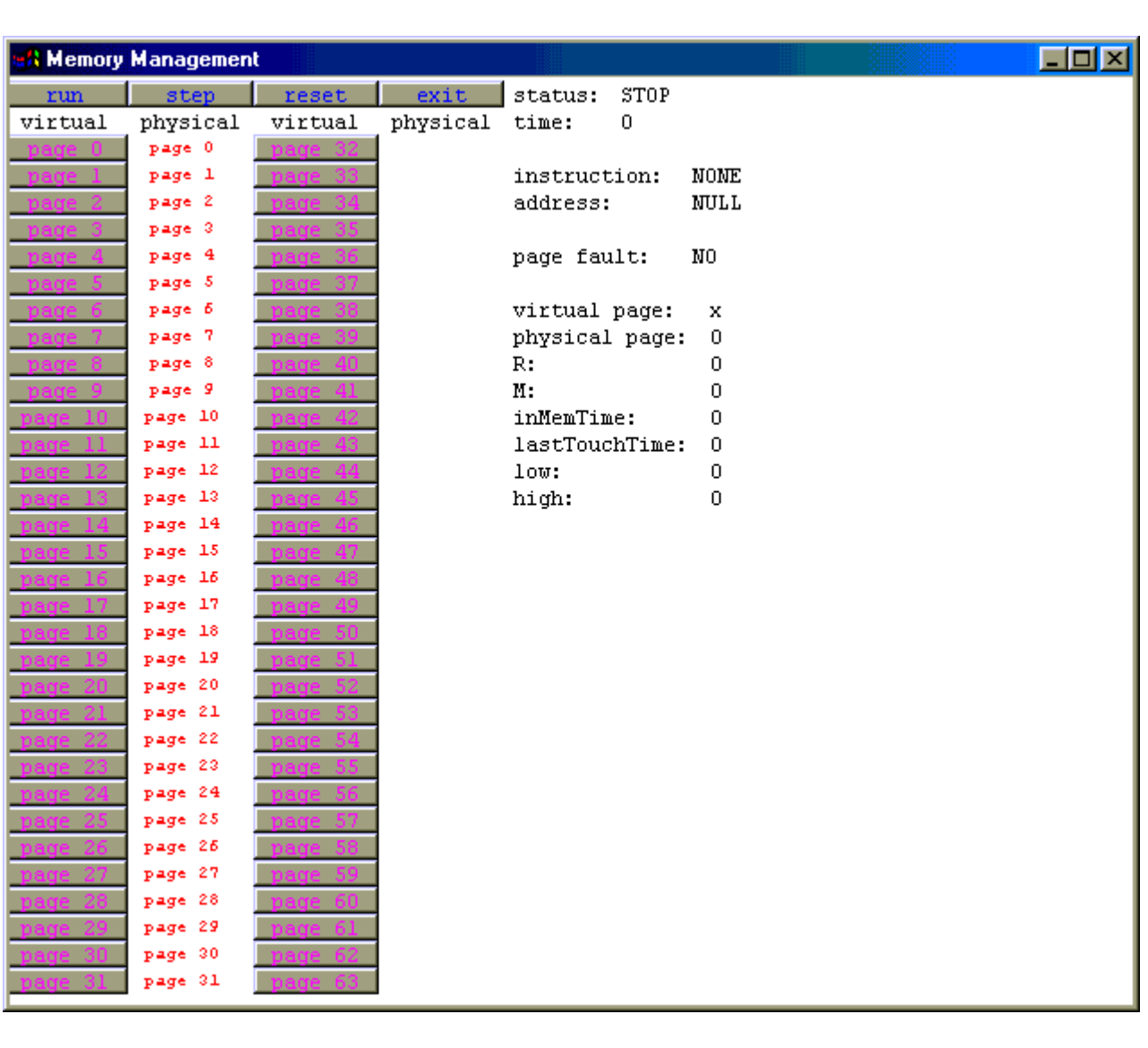
Running the Simulator

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, 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 at the left, and an informational display at the right.



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 *n*	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 *n* 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 preceeded 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 100000000000000
READ bin 100000000000000
WRITE bin 110000000000001
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*
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 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.
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).

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
```

Test

To test the program, enter the following command line.

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

The program will display a window allowing you to run the simulator. When the window presents itself, click on the Run button. You should see the program "execute" 7 memory operations, about one per second. When the simulation completes, click the Exit button.

The memory operation commands are read from a file called "commands", and the initial configuration and various options are specified in the file "memory.conf". The program also produces a log file called "tracefile" in the working directory.

The "commands" file looks something like this:

```
// 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 random
```

If things are working correctly, the "tracefile" should look something like this:

```
READ 4 ... okay
READ 13 ... okay
WRITE 3acc32 ... okay
READ 10000000 ... okay
READ 10000000 ... okay
WRITE c0001000 ... page fault
WRITE 1ff82cdc ... okay
```

The program and its input and output files are described more fully in the *MOSS Memory Management Simulator*.

Please send suggestions, corrections, and comments to Ray Ontko (*rayo@ontko.com*).

Last updated: July 28, 2001