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

Purpose

This document provides instructions for the installation of the MOSS Deadlock 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 Deadlock Simulator and this installation guide were written by Ray Ontko (*rayo@ontko.com*).

Requirements

The following software components are required to install and use the MOSS Deadlock 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/deadlock").

   ```
   $ cd
   $ mkdir moss
   $ cd moss
   $ mkdir deadlock
   $ cd deadlock
   ```

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

3. Expand the compressed tar archive.

   ```
   $ tar -xzf deadlock.tgz
   ```

   or

   ```
   $ gunzip deadlock.tgz
   $ tar xf deadlock.tar
   ```

Files

The directory should now contain the following files:

Files	*Description*
deadlock.tgz	Compressed tar archive which contains all the other files.
Command.java CommandParser.java ControlPanel.java DatFilenameFilter.java DeadlockManager.java Kernel.java OptionsDialog.java Process.java ProcessesDialog.java ProcessesPanel.java Resource.java ResourcesDialog.java ResourcesPanel.java deadlock.java	Java source files (*.java)
Command.class CommandParser.class ControlPanel.class DatFilenameFilter.class DeadlockManager.class Kernel.class OptionsDialog.class Process.class ProcessesDialog.class ProcessesPanel.class Resource.class ResourcesDialog.class ResourcesPanel.class deadlock.class	Compiled Java class files (*.class)
a0.dat a1.dat b0.dat b1.dat	Sample input files
install_unix.html install_windows.html user_guide.html user_guide_1.gif	Documentation and associated images
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 messages, 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 Deadlock Simulator
User Guide

Purpose

This document is a user guide for the MOSS Deadlock 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 Deadlock Simulator and this user guide were written by Ray Ontko (*rayo@ontko.com*).

Introduction

The deadlock simulator illustrates multiple processes competing for one or more resources to investigate the nature and causes of deadlock conditions and how they can be avoided, detected, and resolved. The simulator includes a graphical user interface that allows the student to step through the "programs" being concurrently "executed" by each of the processes and see which processes are blocked by which resources. A typical student lab exercise might require students to write simple "programs" for the simulator to investigate different kinds of resource contention conditions. More advanced students might write a deadlock manager (in Java) based on the template provided, and test it using "programs" of their own design.

Overview

The deadlock simulator performs the following steps as part of the simulation:

- It creates a specified number of simulated processes.
- For each process, it reads a command file of actions to be performed by that process. Each action is one of the following:
  - compute for a specified length of time
  - request an instance of a specified resource
  - free an instance of a specified resource
  - end or halt the process
- It creates a specified number of instances for each of several resources.
- It "executes" the simulation by considering the commands for each process, one at a time, and either allowing the process to execute, granting it an instance of a requested resource, blocking the process until a requested is available, or ending a process.
- As the execution proceeds, the display is updated to reflect the status of each process, and the number of available instances of each resource.
- At the end of each cycle of execution, the simulator writes a "log" message indicating the time since the beginning of the simulation, the number of available instances of each resource, the number of blocked processes, etc.
- The user may "step" through the simulation to view the action at each cycle, or may "run" the simulation to completion.
- When all processes are halted, the simulation stops.

Running the Simulator

To run the program, enter the following command line.

```
$ java deadlock a 2 1 >a.log
```

In this example,

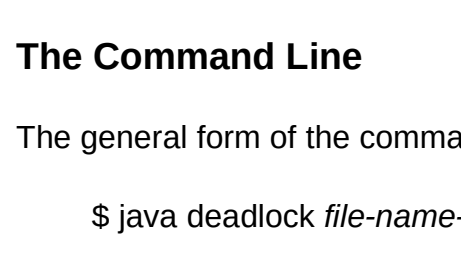
a
: is the prefix for the names of the files containing the commands, (the actual names of the files are "a0.dat", and "a1.dat"),

2
: is the number of processes to be created,

1
: is the number of instances to create for the first resource, and

a.log
: is the name of the output file.

The program will display a window allowing you to run the simulator. You will notice a row of command buttons across the top, and an informational display below. The left side of the information display lists the resources and the number of available instances for each, and the right side lists the processes and the current status for each.



Typically you will use the step button to execute a cycle of the simulation and observe the effect on the resources and processes. When you're done, quit the simulation using the exit button.

The Command Line

The general form of the command line is

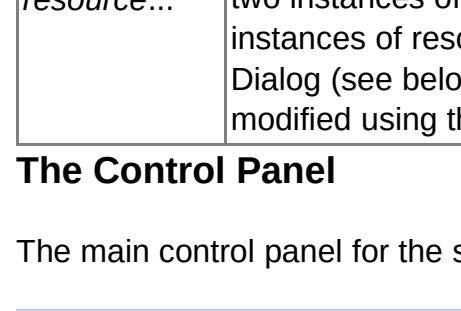
$ java deadlock *file-name-prefix initial-number-of-processes initial-available-for-resource* ...

where

Parameter	*Description*
file-name-prefix	Specifies the name prefix for the process command files. The default command file names are generated from this prefix, followed by the number of the process, followed by ".dat" (e.g, "a0.dat", "a1.dat" if "a" is the prefix). The actual names of the files may be entered or modified in the Processes Dialog (see below).
initial-number-of-processes	Specifies the number of processes to create for the simulation. This should be a non-negative number, usually greater than one. This number may also be entered or modified using the Options Dialog (see below).
initial-available-for-resource...	Specifies the initial number of instances available for each resource. This should be a sequence of non-negative numbers. For example, "2 1 2 0" indicates that there are four resources, and there are initially two instances of resource 0, one instance of resource 1, two instances of resource 2, and zero instances of resource 3. The *number* of resources may also be entered or modified using the Options Dialog (see below). The initial number of instances *available* for each resource may be entered or modified using the Resources Dialog (see below).

The Control Panel

The main control panel for the simulator includes a row of command buttons, and an informational display.



The buttons:

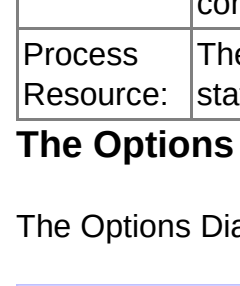
Button	*Description*
run	runs the simulation to completion. Note that the simulation pauses and updates the screen between each step.
stop	stops the simulation if it is running. This button is only active if the run button has been pressed.
step	runs a single setup of the simulation and updates the display.
reset	initializes the simulator and starts from the initial values for each process and resource.
options	allows you to change various options for the simulator, including the number of resources and the number of processes.
resources	allows you to change the configuration for each resource, including the initial and current number of instances available for each resource.
processes	allows you to change the configuration for each process, including current state and the name of the command file for that process.
exit	exits the simulation.

The informational display:

Field	*Description*
Time:	number of "milliseconds" since the start of the simulation.
Resource Id:	A number which identifies the particular resource. Resources are numbered starting with zero.
Resource Available:	The number of instances available for the particular resource. This is a non-negative number.
Process Id:	A number which identifies the particular process. Processes are numbered starting with zero.
Process State:	The current state of the process. This may be U (unknown), C (computable), W (waiting), or H (halted). At the beginning of the simulation, all processes have U status. While a process is computable, it has a C status. If it requests a resource which is unavailable, it enters W status until the resource becomes available. When a process has completed all the commands in its command file or performs a halt command, it enters H status.
Process Resource:	The resource for which this process is waiting, if any. This field only has a value if the process is in W status.

The Options Dialog Box

The Options Dialog Box allows you to set general options for the simulator.

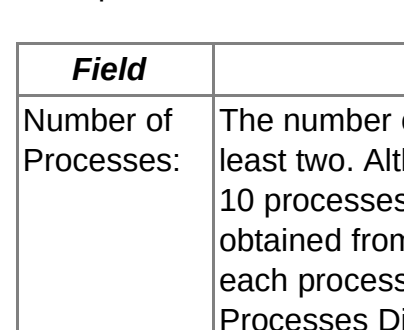


The options:

Field	*Description*
Number of Processes:	The number of processes to use in the simulation. This should be a non-negative number, usually at least two. Although the program does not enforce a limit, you may not be able to view more than about 10 processes on the informational display on your display screen. The initial value for this option is obtained from the second parameter on the command line, or zero, if not specified. Keep in mind that each process should have a process command file. To set properties for individual processes, use the Processes Dialog (see below).
Number of Resources:	The number of resources available in the simulation. This should be a non-negative number, usually at least one. Although the program does not enforce a limit, you may not be able to view more than about 10 resources on the informational display on your display screen. The initial value for this option is obtained from the number of initial instances for each resource specified on the command line (see above), or zero, if none are specified. This number should be one more than the largest resource number mentioned in any of the process command files for the simulation. To set properties for individual resources, use the Resources Dialog (see below).
Milliseconds per step:	The number of real-time milliseconds to pause between each cycle of the simulator in "run" mode. This is the pause between cycles when you hit the run button. The default value is 1000 milliseconds, or, one second.

The Processes Dialog Box

The Processes Dialog Box allows you to enter or modify properties for each process.

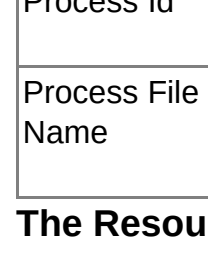


The process properties:

Field	*Description*
Number of Processes:	The number of processes in the simulation. To change this value, use the Options Dialog (see above).
Process Id	The id number for the process. These numbers are used to identify each process and are assigned by the simulator, starting with zero. These numbers cannot be changed.
Process File Name	The name of the file from which process commands are read. This may be any valid filename. For convenience, there is a choose button which allows you to browse the file system to choose the file. By default, the name is the prefix string, followed by the process number, followed by ".dat".

The Resources Dialog Box

The Resources Dialog Box allows you to enter and modify properties for each resource.



The resource properties:

Field	*Description*
Number of resources:	The number of resources available in the simulation. To change this value, use the Options Dialog (see above).
Resource Id	The id number assigned to the resource. This number is used to identify the resource and is assigned by the simulator and cannot be changed. This is the number which appears in the R (request resource) and F (free resource) commands in the process command files.
Resource Initial	The initial number of available instances of the resource. This number is used when the simulator starts or is reset.
Resource Current	The current number of available instances of the resource. This number may be changed during the simulation to see the effect it may have on processes waiting for the resource.

The Process Command Files

The process command files for the simulator specifies a sequence operations to be performed by the process or processes which use the file. There are four operations defined C (compute), R (request resource), F (free resource) and H (halt).

Operation	*Description*
C *msec*	Compute for the specified number of milliseconds (cycles).
R *resource-id*	Request an instance of the specified resource. If none are available, block the process until the resource becomes available. The resource id should be a non-negative number *less than* the total number of resources available.
F *resource-id*	Free an instance of the specified resource. This is usually a resource that was previously requested by the process. The resource id should be a non-negative number *less than* the total number of resources available.
H	Halt the process. This is usually the last operation in the file. Any commands which follow it in the file are ignored. Any file that does not end with this operation is implicitly halted.

Sample Process Command Files

The "a0.dat" input file looks like this:

```
/*
a0.dat

The "a" collection of process data files is meant to simulate
two processes competing for a single resource.  If you run
the simulator with one resource available, one of the processes
will block until the other is done using the resource.
*/
C 10    // compute for 10 milliseconds
R 0     // request resource 0
C 10    // compute for 10 milliseconds
F 0     // free resource 0
H       // halt process
```

Note that the "a1.dat" file is identical. In other words, both files request the same resources at approximately the same time.

The Output File

The output file contains a log of the simulation since the simulation started.

The output file contains one line per cycle executed. The format of each line is:

time = *t* available = *r0 r1* ... *rn* blocked = *n*

where

t
: is the number of milliseconds since the start of the simulation,

r_i
: is the number of available instances of each resource, and

n
: is the number of blocked processes.

Sample Output

The output file "a.log" looks something like this:

```
time = 0 available = 1 blocked = 0
time = 1 available = 1 blocked = 0
time = 2 available = 1 blocked = 0
time = 3 available = 1 blocked = 0
time = 4 available = 1 blocked = 0
time = 5 available = 1 blocked = 0
time = 6 available = 1 blocked = 0
time = 7 available = 1 blocked = 0
time = 8 available = 1 blocked = 0
time = 9 available = 1 blocked = 0
time = 10 available = 0 blocked = 1
time = 11 available = 0 blocked = 1
time = 12 available = 0 blocked = 1
time = 13 available = 0 blocked = 1
time = 14 available = 0 blocked = 1
time = 15 available = 0 blocked = 1
time = 16 available = 0 blocked = 1
time = 17 available = 0 blocked = 1
time = 18 available = 0 blocked = 1
time = 19 available = 0 blocked = 1
time = 20 available = 0 blocked = 0
time = 21 available = 0 blocked = 0
time = 22 available = 0 blocked = 0
time = 23 available = 0 blocked = 0
time = 24 available = 0 blocked = 0
time = 25 available = 0 blocked = 0
time = 26 available = 0 blocked = 0
time = 27 available = 0 blocked = 0
time = 28 available = 0 blocked = 0
time = 29 available = 0 blocked = 0
time = 30 available = 1 blocked = 0
```

In this example, the simulation runs for a total of 30 "milliseconds" and then halts. During the simulation, all processes are computable for 10 milliseconds. During the next 10 milliseconds, the one instance of the resource is allocated to one process, while the other process is blocked. During the final 10 milliseconds, the first process frees the resource, but it is immediately allocated by the second process, which then continues to compute, unblocked, to the end of the simulation.

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

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

Last updated: July 29, 2001