LISTING OF CLAIMS

The claims of the '720 Patent are as follows.

1. A system for dynamic preloading of classes through memory space cloning of a master runtime system process, comprising:

A processor;

A memory

a class preloader to obtain a representation of at least one class from a source definition provided as object-oriented program code;

a master runtime system process to interpret and to instantiate the representation as a class definition in a memory space of the master runtime system process;

a runtime environment to clone the memory space as a child runtime system process responsive to a process request and to execute the child runtime system process; and

a copy-on-write process cloning mechanism to instantiate the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process, and to defer copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process.

2. A system according to claim 1, further comprising:

a cache checker to determine whether the instantiated class definition is available in a local cache associated with the master runtime system process.

3. A system according to claim 2, further comprising:

a class locator to locate the source definition if the instantiated class definition is unavailable in the local cache.

4. A system according to claim 1, further comprising:

a class resolver to resolve the class definition.

5. A system according to claim 1, further comprising:

at least one of a local and remote file system to maintain the source definition as a class file.

6. A system according to claim 1, further comprising:

a process cloning mechanism to instantiate the child runtime system process by copying the memory space of the master runtime system process into a separate memory space for the child runtime system process.

7. A system according to claim 1, wherein the master runtime system process is caused to sleep relative to receiving the process request.

8. A system according to claim 1, wherein the object-oriented program code is written in the Java programming language.

9. A system according to claim 8, wherein the master runtime system process and the child runtime system process are Java virtual machines.

10. A method for dynamic preloading of classes through memory space cloning of a master runtime system process, comprising:

executing a master runtime system process;

obtaining a representation of at least one class from a source definition provided as object-oriented program code;

interpreting and instantiating the representation as a class definition in a memory space of the master runtime system process; and

cloning the memory space as a child runtime system process responsive to a process request and executing the child runtime system process; wherein cloning the memory space as a child runtime system process involves instantiating the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process; and wherein copying references to the memory space of the master runtime system process defers copying of the memory space of the master runtime system

process until the child runtime system process needs to modify the referenced memory space of the master runtime system process.

11. A method according to claim 10, further comprising:

determining whether the instantiated class definition is available in a local cache associated with the master runtime system process.

12. A method according to claim 11, further comprising:

locating the source definition if the instantiated class definition is unavailable in the local cache.

13. A method according to claim 10, further comprising:

resolving the class definition.

14. A method according to claim 10, further comprising:

maintaining the source definition as a class file on at least one of a local and remote file system.

15. A method according to claim 10, further comprising:

instantiating the child runtime system process by copying the memory space of the master runtime system process into a separate memory space for the child runtime system process.

16. A method according to claim 10, further comprising:

causing the master runtime system process to sleep relative to receiving the process request.

17. A method according to claim 10, wherein the object-oriented program code is written in the Java programming language.

18. A method according to claim 17, wherein the master runtime system process and the child runtime system process are Java virtual machines.

19. A computer-readable storage medium holding code for performing the method according to claim 10.

20. An apparatus for dynamic preloading of classes through memory space cloning of a master runtime system process, comprising:

A processor;

A memory

means for executing a master runtime system process;

means for obtaining a representation of at least one class from a source definition provided as object-oriented program code;

means for interpreting and means for instantiating the representation as a class definition in a memory space of the master runtime system process; and

means for cloning the memory space as a child runtime system process responsive to a process request and means for executing the child runtime system process; wherein the means for cloning the memory space is configured to clone the memory space of a child runtime system process using a copy-on-write process cloning mechanism that instantiates the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process and that defers copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process.

21. A system according to claim 1, further comprising:

a resource controller to set operating system level resource management parameters on the child runtime system process.

22. A method according to claim 10, further comprising:

setting operating system level resource management parameters on the child runtime system process.

LISTING OF APPENDICES

The following Appendices are submitted herewith.

Appendix	Description
A	Declaration of Dr. Benjamin Goldberg ("Goldberg Declaration")
B	"CDC Porting Guide for the Sun Java Connected Device Configuration Application Management System," Version 1.0, Sun Microsystems, Inc., November 2005 ("Porting Guide")
C	"CDC Runtime Guide for the Sun Java Connected Device Configuration Application Management System," Version 1.0, Sun Microsystems, Inc., November 2005 ("Runtime Guide")
D	"Anatomy and Physiology of an Android, Google I/O 2008," by Patrick Brady, *http://sites.google.com/site/io/anatomy-physiology-of-an-android/Android-Anatomy-GoogleIO.pdf* ("Android Presentation," last visited July, 5, 2011)
E	"Dalvik Virtual Machine Internals, Google I/O 2008," by Dan Bornstein, *http://sites.google.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf* ("Dalvik Presentation," last visited July 5, 2011)
F	"Bug ID: 4416624 multiple JVM runtimes do not share memory between themselves," *http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4416624* ("Shared Memory Blog," last visited July 5, 2011)
G	"Bug ID: 4469557 Faster startup/reduced footprint for subsequent VMs," *http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=44679557* ("Reduced Footprint Blog," last visited July 1, 2011)
H	"Complaint for Patent and Copyright Infringement," *Oracle America, Inc. v. Google, Inc.*, N.D. Cal., Case No. cv10-03561 ("Complaint")
I	"Zygote" reference page, *http://59.61.88.234/android-help/reference/dalvik/system/Zygote.html* ("Zygote," last visited July 5, 2011)
J	"Evidence of Copying by Google of US 7,426,720" Chart ("Copy Chart")
K	"Multitasking VMs: More Performance, Less Memory," by Kyle Buza, et al., JavaOne Conference, June 2005 ("JavaOne Presentation")
L	"Cloneable JVM: A New Approach to Start Isolated Java Applications Faster," by Kiyokuni Kawachiya, et al., *VEE'07*, June 2007 ("Kawachiya Paper")