A method and system to provide concurrent access to a software object. At least some of the illustrative embodiments may be a method comprising creating an object having a read-write field and read-only field, accessing the read-write field by a first thread, duplicating the read-write field to the read-only field, and then concurrently accessing the read-write field by the first thread and the read-only field by the second thread.
FIG. 1

100

114

DISPLAY

112

BYTECODES

COMPILER

108

JVM

102

JSM

MPU

106

MEMORY

FIG. 2

@01

204

@02

210

NULL

208

NULL

212

NULL

FIG. 3

300

START

302

CREATE OBJECT WITH A READ-WRITE FIELD AND A READ-ONLY FIELD

304

ACCESS THE READ-WRITE FIELD BY A FIRST THREAD

306

DUPLICATE THE READ-WRITE FIELD TO THE READ-ONLY FIELD

308

ACCESS READ-WRITE FIELD BY THE FIRST THREAD

310

ACCESS READ-ONLY FIELD BY A SECOND THREAD

312

END
METHOD AND SYSTEM PROVIDE CONCURRENT ACCESS TO A SOFTWARE OBJECT

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of European Patent Application No. 04291918.3, filed Jul. 27, 2004, incorporated by reference herein as if reproduced in full below.

BACKGROUND OF THE INVENTION

[0002] 1. Technical Field of the Invention

[0003] The present invention relates to processors, and more particularly to processors executing software streams that each accesses a common object.

[0004] 2. Background Information

[0005] Java™ is a programming language that, at the source code level, is similar to object oriented programming languages such as C++. Java™ language source code is compiled into an intermediate representation based on a plurality “bytecodes” that define specific tasks. In some implementations, the bytecodes are further compiled to machine language for a particular processor.

[0006] Many high level languages such as Java allow the programmer to dynamically allocate and deallocate memory. Dynamically allocated memory may, however, become unreachable when there is no chain of references to the memory in question from the root object. Unreachable objects that the programmer fails to deallocate may be referred to as “garbage.” To address unreachable memory, some systems implement an automatic reclamation of unreachable memory known as “garbage collection”. The garbage collection software runs periodically and traces through the various software objects created by application programs looking for objects that are garbage. The garbage collection program “collects” the garbage objects and returns their memory to the heap for further allocation.

[0007] The garbage collection software thus traces through each object, even objects that are in active use by the application programs. Confinement as to memory locations is possible as between the application programs and the garbage collection software, thus slowing the application programs. In some cases, the application programs are stopped while garbage collection takes place.

SUMMARY

[0008] The problems noted above are solved in large part by a method and system to provide concurrent access to a software object. At least some of the illustrative embodiments may be a method comprising creating an object having a read-write field and read-only field, accessing the read-write field by a first thread, duplicating the read-write field to the read-only field, and then concurrently accessing the read-write field by the first thread and the read-only field by a second thread.

NOTATION AND NOMENCLATURE

[0009] Certain terms are used throughout the following description and claims to refer to particular system components. As one skilled in the art will appreciate, semiconductor companies may refer to a component by different names. This document does not intend to distinguish between components that differ in name but not function. In the following discussion and in the claims, the terms “including” and “comprising” are used in an open-ended fashion, and thus should be interpreted to mean “including, but not limited to . . . ”. Also, the term “couple” or “couples” is intended to mean either an indirect or direct connection. Thus, if a first device couples to a second device, that connection may be through a direct connection, or through an indirect connection via other devices and connections.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] For a more detailed description of the preferred embodiments of the present invention, reference will now be made to the accompanying drawings, wherein:

[0011] FIG. 1 shows a diagram of a system in accordance with embodiments of the invention;

[0012] FIG. 2 shows a memory having a plurality of objects in accordance with embodiments of the invention;

[0013] FIG. 3 shows a method in accordance with embodiments of the invention; and

[0014] FIG. 4 illustrates graphically operation of at least some embodiments of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0015] The following discussion is directed to various embodiments of the invention. Although one or more of these embodiments may be preferred, the embodiments disclosed should not be interpreted, or otherwise used, as limiting the scope of the disclosure, unless otherwise specified. In addition, one skilled in the art will understand that the following description has broad application, and the discussion of any embodiments is meant only to be illustrative of those embodiments, and not intended to intimate that the scope of the disclosure is limited to those embodiments.

[0016] Moreover, the various embodiments of the invention were developed in the context of Java programs and their associated garbage collection programs, and so the description is related to that developmental context; however, the methods and systems find applicability outside the Java environment, and so the description should not be construed as a limitation to the breadth of the claims.

[0017] FIG. 1 shows a system 100 in accordance with embodiments of the invention. As shown, the system may comprise at least two processors 102 and 104. Processor 102 may be referred to for purposes of this disclosure as a Java Stack Machine (“JSM”) and processor 104 may be referred to as a Main Processor Unit (“MPU”). System 100 may also comprise memory 106 coupled to both the JSM 102 and MPU 104. At least a portion of the memory 106 may be shared by both processors, and if desired, other portions of the memory 106 may be designated as private to one processor or the other. System 100 also comprises a Java Virtual Machine (“JVM”) 108, a compiler 110, and a display 114. The JVM 108 may comprise a combination of software and hardware. The software may comprise the compiler 110 and the hardware may comprise the JSM 102. The JVM may
comprise a class loader, bytecode verifier, garbage collector, and a bytecode interpreter loop to interpret the bytecodes that are not executed on the JSR processor 102. Other components (not specifically shown) may be included as desired for various applications.

[0018] Java language source code is converted or compiled to a series of bytecodes 112, with each individual one of the bytecodes referred to as an “opcode.” Bytecodes 112 may be provided to the JVM 108, possibly compiled by compiler 110, and provided to the JSR 102 and/or MPU 104 for execution. In accordance with some embodiments of the invention, the JSR 102 may execute at least some Java bytecodes directly. When appropriate, however, the JVM 108 may also request the MPU 104 to execute one or more Java bytecodes not executed or executable by the JSR 102. In addition to executing compiled Java bytecodes, the MPU 104 also may execute Java garbage collection as well as non-Java instructions. The MPU 104 may thus also host an operating system (“O/S”) (not specifically shown) which performs various functions such as system memory management, system task management that schedules the software aspects of the JVM 108 and most or all other native tasks running on the system, management of the display 114, and receiving input from input devices (not specifically shown). Java code, whether executed on the JSR 102 or MPU 104, may be used to perform any one of a variety of applications such as multimedia, games or web based applications in the system 100, while non-Java code, which may comprise the O/S and other native applications, may still run on the system on the MPU 104.

[0019] In accordance with embodiments of the invention, Java or other programs executing on one or both of the JSR 102 and MPU 104 allocate memory, and within the allocated memory create objects. Each object has one or more internal components known as fields. FIG. 2 illustrates a first object 200 (in this case the root object “#001”) allocated within a memory 202. In particular, the object 200 comprises a field 204 that contains a reference to other objects, which in this case is a reference to object 206 within the same memory 202. Object 206 also has a reference, but since in this illustrative case the object 208 is the last object in the series, its field 208 contains a NULL reference. In accordance with embodiments of the invention, application programs (such as Java programs) access the fields 204 and 208 in a read-write fashion. In the related art, garbage collection programs (which may execute on a different processor than the application programs) also access the fields 204 and 208 (in a read-only fashion) in the process of tracing to identify and free unreachable objects. Access to these fields may thus require consistency management by applying a cache coherency protocol if the application program and garbage collection program run simultaneously, slowing each programs access. In further related art implementations, the application program may be stopped while the garbage collection program runs, thus imposing a performance penalty in the application program to implement the garbage collection process.

[0020] Still referring to FIG. 2, in accordance with embodiments of the invention, each object is created with an additional field. For example, object 200 has an additional field 210, and object 206 has an additional field 212. In broad terms, and referring to object 200, the additional field 210 is periodically updated with data from the primary field 204. Likewise for object 206, the additional field 212 is periodically updated with the data from primary field 208. In this way, an application program may access the primary field in each object, and other programs, such as garbage collection programs, that need access to the data may access the addition field, thus avoiding contention.

[0021] Although the additional fields may be updated at any suitable frequency, in accordance with at least some embodiments the updating of the additional fields may take place just prior to running programs that need access to the data, such as the garbage collection programs. Thus, the application programs may continue to access the primary fields in a read-write fashion, and the illustrative garbage collection program may perform its task with respect to the additional fields. It is again noted that garbage collection programs are merely illustrative of a family of programs that may need access to shared data, but for which strong consistency of the data is not required. Copying the data from the primary fields the additional fields may also be referred to as fixing a memory view at a particular point in time.

[0022] FIG. 3 illustrates a method in accordance with embodiments of the invention. In particular, the illustrative method starts (block 300) and proceeds to creating an object with a read-write field and a read-only field (block 302). In accordance with embodiments operating in a Java environment, creation of the object may be accomplished by the JVM 108 (FIG. 1). Regardless of the precise operating environment, a first thread (e.g., an application program written in Java) accesses the read-write field of the object (block 304). At some point, a second thread, possible executed on a second processor (e.g., MPU 104 of FIG. 1) may want access to the data of the read-write field, but may not want to hamper access to that field by the first thread. Stated otherwise, the memory view for the second thread may need to be fixed. Thus, the data from the read-write field may be copied to the read-only field (block 306). In accordance with embodiments operating in a Java environment, duplicating of the read-write field to the read-only field may take place by the JVM 108. After duplication (again block 306), the first thread accesses the read-write field (block 308), and the second thread may concurrently access the read-only field (block 310). Thereafter, the process ends (block 312).

[0023] FIG. 4 illustrates graphically operation of at least some embodiments of the invention. In particular, FIG. 4 illustrates a memory area 400 into which a first object 402 is created. Being initially the only object, the read-write field 404 and the read-only field 406 contain NULL references. At some point thereafter, a second object 408 is created, and the read-write field of the first object is modified to point to the second object. At this point, the read-only field 406 of the first object and both fields of the second object each contain a NULL reference. Further consider that between creating the second object and creating a third object a memory view fix is run, illustrated by vertical line 414. As discussed above with respect to the illustrative method of FIG. 3, running a memory fix involves copying the read-write field of each object to the read-only field of the same object. These copies are illustrated in FIG. 4 by line 416 for the first object 402, and line 418 for the second object 408. In the illustrative situation of FIG. 4, the read-write field of the first object 402 is modified to point to the third object 412, and the third
Thus, the memory view present just before the memory fix illustrated by line 414 is held in the read-only fields of the first and second objects 402 and 408 respectively after the memory fix.

Now further consider that a second memory fix is applied, as illustrated by vertical line 420, and thus the read-write field of each object is copied to the read-only field, as illustrated by lines 422, 424 and 426 for each of the objects 402, 408 and 412 respectively. Also, after the memory fix illustrated by line 420 read-write field of object 402 is modified to have a NULL reference (which as we shall see acts to make objects 408 and 412 unreachable, and therefore garbage). Finally consider that yet another memory fix is run as illustrated by vertical line 428, where again the contents of each objects’ read-write field is copied to its read-only field. After each memory fix, a second thread, such as a garbage collection program, may operate on the read-only fields of the objects without interference with a first thread operating on the read-write fields of those same objects, and the fact there is not a strong consistency is immaterial, especially for the illustrative garbage collection process as the second thread

Analyzing FIG. 4 from the standpoint of an illustrative garbage collection process, after the first memory fix (vertical line 414), the read-only field of the first object 402 points to the second object, and the third object is new (has no previous read-write state), and thus no garbage objects are present. After the second memory fix (vertical line 420), as far as the illustrative garbage collection program can tell the first object 402 points to the third object 412, and the third object points to the second object 408, and so again no garbage objects are present. Finally, after the third memory fix (vertical line 428), while the first object is still the root object, neither the second object 408 nor the third object 412 are reachable, and thus both the second and third objects are garbage, and their memory areas may be returned to the heap.

While the various embodiments of the invention have been shown and described, modifications thereof can be made by one skilled in the art without departing from the spirit and teachings of the invention. The embodiments described herein are illustrative only, and are not intended to be limiting. Many variations and modifications of the invention disclosed herein are possible and are within the scope of the invention. Each and every claim is incorporated into the specification as an embodiment of the present invention.

What is claimed is:

1. A method comprising:
   creating an object having a read-write field and read-only field;
   accessing the read-write field by a first thread;
   duplicating the read-write field to the read-only field; and
   then concurrently accessing the read-write field by the first thread and the read-only field by a second thread.

2. The method as defined in claim 1 wherein concurrently accessing further comprises accessing the read-write field of the object by the first thread executed on a first processor, and accessing the read-only field by the second thread executed on a second processor.

3. The method as defined in claim 2 wherein concurrently accessing further comprises accessing the read-write field by an application program, and accessing the read-only field by a garbage collection program.

4. A computer readable medium storing a first program that, when executed by a processor, performs a method comprising:
   creating a software object having a first field and a second field, the first field accessed by a second program;
   periodically stopping execution of the second program and duplicating the first field to the second field; and then
   accessing the second field by the first program simultaneously with accessing of the first field by the second program.

5. The computer readable medium as defined in claim 4 wherein accessing further comprises accessing the second field by the first program executing on a first processor, and accessing the second field by the second program executing on a second processor.

6. The computer readable medium as defined in claim 5 wherein creating further comprises creating the software object in a cache memory accessible to both the first and second processors.

7. The computer readable medium as defined in claim 4 wherein accessing further comprises accessing the second field by the first program being a user program, and accessing the second field by the second program being a program to identify and free unused software objects.

8. A system comprising:
   a first processor;
   a second processor; and
   a memory coupled to the first and second processor;

   wherein the memory is configurable to store a software object; and

   wherein a first process executing on the first processor writes to a first field of the object, and when a second process needs the access to the object, the first field is copied to a second field, and thereafter the first and second processes have the capability to access the object simultaneously.

9. The system as defined in claim 9 wherein the first process executed on the first processor is an application program, and the second processes executed on the second processor is configured to identify unused software objects.

10. The system as defined in claim 9 wherein the memory is a cache memory.

11. The system as defined in claim 9 wherein the first process has read-write access to the first field, and wherein the second process has only read access to the second field.

12. A method comprising:
   operating an application program accessing a plurality of software objects, each software object having a read-write field;
   ceasing operation of the application program; and then
   copying, within each object, the read-write field to a read-only field; and
accessing a read-write field of a first of the plurality of software objects by the application program, and simultaneously accessing the read-only field of the first of the plurality of software objects by a second program.

13. The method as defined in claim 12 wherein accessing further comprises accessing the read-only field of the first of the plurality of software objects by a program configured to identify unused software objects.

14. The method as defined in claim 12 wherein accessing further comprises accessing by the application program executed on a first processor, and the second program executed on a second processor.

15. The method as defined in claim 12 wherein accessing further comprises accessing by the application program executed as a first thread on a processor, and accessing by the second program as a second thread on the processor.