Simulation of Real-Time Scheduling Algorithms in Object Oriented Systems

Goran Jakovljević, Zvonimir Rakamarić, Domagoj Babić
Faculty of electronic engineering and computing, Zagreb 2002.
goran.jakovljevic@fer.hr, {zrakamar, dbabic}@rasip.fer.hr

Abstract. *This document describes authors' research in application of object-oriented language (Java) in real-time systems. It describes advantages and disadvantages of Java, and gives a critic overview of necessary modifications to make Java an acceptable choice for real-time systems. Because of inherent constraints of existing run-time environments, this paper deals only with a part of complete problem, which was possible to simulate on existing Java platform. We have developed modular aperiodic scheduling algorithm simulator. Simulator implements EDF scheduling algorithm, but it can easily be extended to support any aperiodic scheduling algorithm.*

Keywords. Real time java, scheduling simulation,

1 Introduction

Real time system is a system that must react to events from its environment with precise time constraints. As a consequence, accurate behavior of such system does not depend only on correctness of computation, but also on correctness in time domain. Delayed reaction to some important event can be useless, even dangerous or catastrophic depending on real time system type (hard real-time, firm real-time, soft real-time).

Development techniques used in the past are based on low level of abstraction like assembler programming and cannot offer short development cycle nor satisfy rigid maintenance demands. By using object-oriented approach to design real-time systems, programmer has more chance to concentrate to the problem that has to be solved. There is a clear difference between problem space and solution space, and programmer is relieved from binding this spaces together. That makes software portability and maintenance easier.

The rest of this paper describes the results of our work. Section 2 gives an overview of real-time systems. Section 3 describes Java in relation to real-time systems. The basic explanation of architecture and function of our simulator is given in section 4. In section 5, we conclude.

1.1 Basic terms

Under process we assume computation executed on central processing unit in sequential manner. Synonyms are task and job, although some authors prefer to distinguish between them.

The set of rules that determines an order of process execution is called scheduling algorithm. Every process in system passes through set of states. Process that can be executed on processing unit is called "active" process. Process that waits for a free CPU time slot is called "ready" process. Currently executing process is called "running" process. Graphical representation is given on Fig. 1.

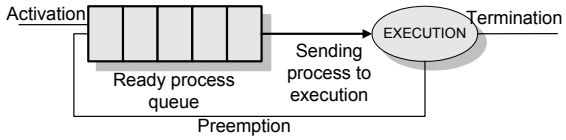

Figure 1. Process states

In dynamic systems it must be possible to replace the currently executing process with new process that is scheduled for execution and has higher priority than the current one. In this case currently executing process is suspended and put into ready queue, and newly arrived process with higher priority is started. This operation is known as preemption. In dynamic real-time systems preemption is important for three reasons:

1. Processes intended to do the interrupt processing are of great importance and it is important to execute them as soon as they enter the system.
2. When processes in one system have highly diverse priorities, preemption makes more efficient process scheduling possible.
3. System responsiveness to events from system environment is much better.

Processes execution constraints according to [1] can be grouped to classes:

1. Time constraints
2. Precedence constraints
3. Mutual exclusion constraints (over shared resources)

Real time process can be characterized by the following parameters:

1. Arrival time Ai – is the time at which a process is ready to execute.
2. Computation time Ci – is the amount of time needed to complete process computation on a given processor without any interruption.
3. Deadline time Di – is the time before which a process should be completed.
4. Start time Si – is the time at which a process starts its execution.
5. Finish time Fi – is the time at which a process finishes its execution.
6. Lateness Li: Li = Fi – Di
7. Laxity Xi: Xi = Di – Ai – Ci

Process parameters are visualized on Fig. 2.

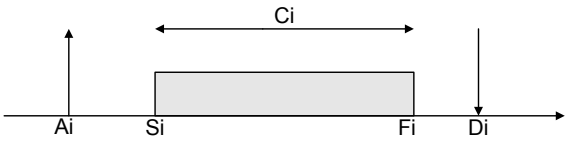

Figure 2. Process parameters

Regarding to time characteristics, processes can be divided into periodic and aperiodic. Periodic processes are composed of endless set of identical activities, which are regularly activated with a constant frequency. Generally, there are both periodic and aperiodic processes in real-time systems.

Precedence constraints bind processes together in a way that one process has to wait for the results of execution of the other process. These relationships between processes are best described by directed acyclic graph.

.

2 Java & real time

One of the most interesting Java advantages is possibility of portable real time software. That advantage comes from the architecture of Java runtime. Besides development of portable software, it is possible to integrate software already written in other languages using Java JNI (Java Native Interface).

Java interpreter interprets compiled Java code (java bytecode) on the host machine. Due to interpretation, Java code run much slower than code compiled specifically for host architecture. To overcome this problem JIT (Just In Time) compilers have been introduced. JIT compilers add more non-determinism to application execution, which is unacceptable for real time systems. One of the possible solutions is to recompile Java bytecode statically to native code, but this constrains portability, one of the Java's most important features.

2.1 Java bytecode structure

Java bytecode [8,10] holds information essential to Java runtime environment for creating and executing Java application. Consequence or rigid requests for portability and security is absence of any kind of device and memory access instructions in bytecodes.

From perspective of real-time system, it is imperative to find out worst-case execution time. Java bytecode mostly consists of instructions that have predictable worst-case execution time, but there are also a few whose worst-case execution time cannot be predicted (lookupswitch and tableswitch). Although percentage of these instructions in bytecode is very small, they put a risk at predictability of hard real time application.

2.2 Embedded concurrent programming support

Java's concurrent programming support is big advantage in development of real time applications. To achieve predictability in real time applications written in Java, Java run time must implement adequate support.

Java real-time threads support only a part of whole thread package defined in POSIX specification. Java threads evolved from Green Thread package written on top of SunOS system version 4.1.3.

Synchronization is one of the basic elements of multitasking environment because it enables threads to coordinate their actions when manipulating shared resources. Main method of synchronization in Java is based on Hoare's monitor. POSIX defines other synchronization primitives like mutex, semaphores, message queues, etc. but Java does not implement that many mechanisms. Other mechanisms can easily be implemented. In real time Java specification semantics of synchronized keyword is expanded to avoid priority inversion problem and wait queues supports defined priorities. "Priority

inheritance" algorithm is suggested as a possible solution of priority inversion.

Scheduling is also a part of standard Java specification that had to be modified. In standard specification in the case of threads competing for the same resource, generally advantage is given to the process with higher priority, but it is also possible for thread of lower priority to get the resource. It is obvious that real time systems need more rigid semantics. Also there are not enough priority levels (only 10). RT Java Specification [12] defines "fixed priority" scheduling protocol and 38 levels of priority.

2.3 Automatic memory management

Automatic memory management relieves programmer from burden of memory allocation and deallocation. Besides, explicit memory allocation and deallocation can be a source of serious and hard to detect errors. In Java new objects are created in memory part called "heap", and deallocation of memory allocated to objects that are not needed any more is done automatically by run-time component called "garbage collector" (GC). Garbage collectors have big disadvantage from the aspect of hard real-time systems – they introduce lateness and unpredictability. In today's Java implementations GC is implemented as a high priority thread which is run when the memory use reaches certain limit. Execution time of "stop and copy" garbage collecting algorithm depends on the heap size, and execution time of "mark and sweep" algorithm depends on quantity of dead objects in heap memory. Consequence of above facts is that the precise moment of GC start cannot be determined. GC run duration is also unknown. For soft-real time systems, multimedia and interactive applications, GC algorithms called incremental collectors (memory is deallocated in small steps) and generation collectors (objects in memory are divided in generations and younger generations are deallocated more often) are more appropriate. These algorithms show good performance in soft-real time systems, but for hard real-time systems no strict time guarantees can be given.

RT Java Specification defines, beside heap memory, additional memory areas that are not deallocated by standard mechanisms. Strict access, assignment and allocation rules prevent dangling references and protect pointer security. Objects allocated in these additional memory areas can contain references to objects in heap memory area. This is a way of making memory areas inaccessible for GC and of insuring that predictability of executing code situated in these areas can be maintained. This code has higher priority than GC thread. Problem of this objects not being deallocated automatically is not important for hard real-time systems because only small amount of code is kept in this areas – that code changes rarely and does not use a lot of memory.

2.4 Dynamic behavior

Java enables code changes that can happen "on the fly" because classes can be inserted in system dynamically. This feature is useful in applications with high flexibility requirements. This feature can be used also in hard real time systems for applications that need to make some changes on the system very quickly and without interruption of running system.

2.5 Java security

Every class file that is thoroughly analyzed and verified before loading what gives Java bases systems very high security. Anyway, dynamic loading has unpredictable timing, so some real-time java systems use static loading [22].

Platform independency has a drawback because system hardware cannot be accessed directly. For embedded applications this is a problem. Applications must have ability to access hardware if they want to implement device drivers, interrupt routines, etc. But Java offers elegant solution in form of JNI (Java Native Interface). This way application parts can be written in C/C++ or some other language of user choice. RT Java Specification defines two classes for raw memory access directly from Java code: **RawMemoryAccess** and **PhysicalMemory**.

3 Real-time scheduling simulator

Real time scheduling simulator is application written completely in Java, that simulates simplified Java real-time thread scheduling. It is a multithreaded simulator that consists of two components:

1. Graphical user interface that displays information about current system state (number and parameters of all processes in

system, processes feasibility, system time and processor usage information).

2. Simulation kernel that accepts new processes in system, schedules threads and tests feasibility of a given thread composition. Objects that present aperiodic processes are generated dynamically and their feasibility is computed. If process is feasible it's being scheduled for execution. Simulated kernel makes dynamic decisions about scheduling based on current system status.

These two components are implemented independently. That means that any of them can be changed without influence to another one. Fig. 3 shows simplified block model of simulation system.

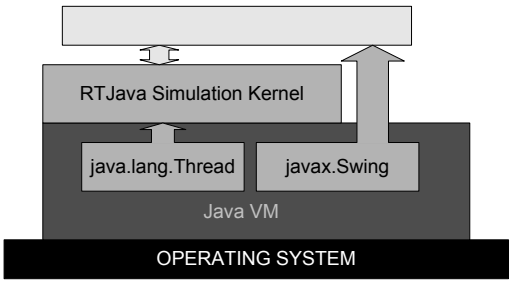

Figure 3. Block model of simulation system

3.1 Description and characteristics of processes in the system

Only aperiodic processes can appear in the system. Maximum number of processes in the system at any moment is 15, and intensity of new process generation can be controlled. **ThreadGenerator** class generates processes with Gauss distribution with its mean value set by user. Relative process parameters are:

1. Process ID – process identification number
2. Deadline – the time until process has to be completed
3. Cost – is the amount of time needed to complete process computation on a given processor without any interruption

Simulator evaluates process feasibility in a given timeframe. Light indicator displays feasibility result for every process (green light is on if process is feasible). Same way of signalization is used to show currently running process.

Every process is represented as **RealTimeThread** object that remembers parameters relevant for simulation. This class also implements utilities for parameter retrieval called by GUI methods. Every process is executed on the virtual processor represented by **ProcessorEngine** class. **ProcessorEngine** takes thread objects from FIFO queue, executes processes and calculates current processor usage. Data about processor usage are forwarded to Graphical user interface and **ProcessInfoPanel** object is responsible for their displaying.

3.2 Scheduling

Scheduling problem deserves great attention. Our simulator accepts only aperiodic processes. In real real-time systems there can be periodic as well as aperiodic processes. Typically periodic processes execute critical control activities with rigid time constraints. Aperiodic processes are typically event driven and can be hard, soft and non-real time. In hybrid system the goal of the system is to execute all hard real-time processes in time and to execute soft real-time processes with the best performance. In this kind of system periodic processes are scheduled based on their fixed priorities (for example Rate Monotonic algorithm schedules processes based on their time periods), while aperiodic processes are scheduled using server (Polling server, Deferrable server, Sporadic server, Slack stealing) or using background scheduler. In our simulator processes are generated dynamically and on-line scheduling algorithm is implemented. EDF (Earliest Deadline First) algorithm was chosen for referent implementation, because it is optimal algorithm regarding maximal lateness. Real time Java simulator supports simple scheduling algorithm change. Complete scheduling logic is defined in the **Comparator** object. In the case of EDF implementation, its subclass **EDFComparator** is used. To implement any other aperiodic scheduling algorithm, it is sufficient to subclass and implement **Comparator** class, which is used for scheduling. Similar mechanism of priority changing is implemented in **Spring** distributed real time system [1]. Illustrative description of this concept is shown on Fig. 4.

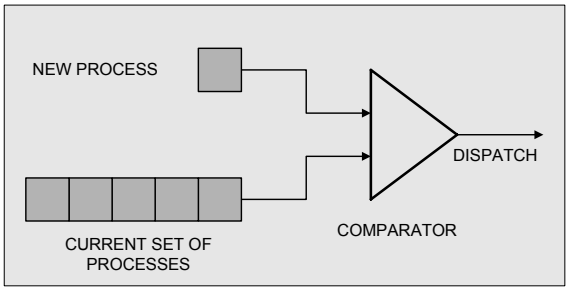

Figure 4. Comparator scheduling logic

Class called **Scheduler** is used in our system to deal with process scheduling.

3.3 System time

For the purpose of measuring system time, coordinating time dependent actions and analyses of time constraints inside system, we created **SystemClock** class. Because time is essential part of real time system, part that is used for making decisions, special attention was dedicated to designing this part of the system. This class is used by many components, so it was necessary to enable concurrent access to this class. Main functions of **SystemClock** class are:

1. Registration of deadline point
2. Signalization of missed deadline
3. Communication with component for graphical displaying of system time. (**TimerPanel** class).

Role of the system clock is essential in real-time systems. Fig. 5 illustrates dependencies inside the system.

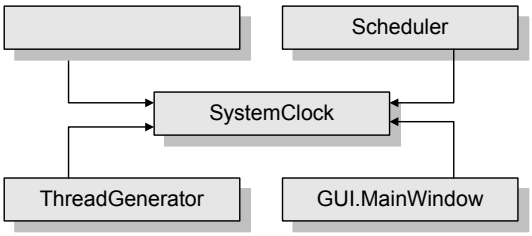

Figure 5. System clock dependencies

3.4 Graphical user interface

Graphical user interface is used to give graphical insight to the simulation system status (currently active processes, feasible processes, processes that have missed deadline, system time, processor utilization) and to assist in changing system parameters. User changeable system parameters are:

- Intensity – mean intensity of process generation. Minimal number or processes is 0 and maximal number is 15. These values are defined in class **SimulationSystemParameters**.
- Cost – mean run time cost of generated processes. In simulated environment this corresponds to the minimal time the process must execute without interruption to complete.
- Deadline – mean deadline value for generated processes. Maximal deadline value is 150 time units, and it is defined in class **SimulationSystemParameters.**

Graphical user interface is organized in 3 basic windows, and they are contained in **MainWindow** class:

1. Process information display (**ThreadsInfoPanel**) is used for adjusting and displaying basic system parameters. Fig. 6 shows screenshot of this window. One can notice lights that indicate current state of all 15 processes in system: active – active panel, feasible – feasible panel and miss deadline – miss deadline panel (indicates that process missed its deadline). Other displayed parameters are process id, cost, deadline and sliders for adjusting intensity, average cost and average deadline.
2. CPU utilization window **Processor-InfoPanel** – shows processor utilization percentage information.
3. System clock window (**TimerPanel**) shows system time.

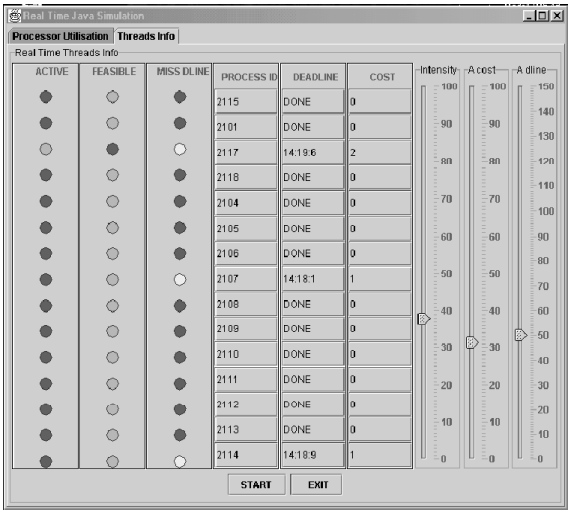

Figure 6. ThreadsInfoPanel window

3.5 Further development

Because of inherent limitation of current Java implementation regarding real time system needs, we simulated only a part of complete problem related to scheduling. Further development of Real-time scheduling simulator should support simulation of both periodic and aperiodic processes, to make better insight into the problem.

4 Conclusion

Java offers many interesting features for programmer of real-time and embedded applications. Many problems regarding real time programming in Java need more research. As a programming language, Java has many advantages over C or C++, but today's Java implementations still have many disadvantages for usage in real-time systems. The most important disadvantages are GC unpredictability, slowness in comparison with C/C++ and the absence of direct memory access instructions.

When programming Web applications, games and interactive applets, Java features like portability and high level of abstraction are desirable. Real-Time Java specification reflects the intention to adjust Java to the needs of real-time applications. Performance of interpreted Java code is bad, but solutions are available in the form of JIT compilers, ahead of time compilers and static linkage of Java classes. Built in support for multithreading enables simple development of multitasking applications. Automatic memory management also represents a big challenge to real time systems.

5 Literature

1. G. C. Buttazzo, Hard Real-Time Computing Systems, Kluwer Academic Publishers, Boston, 2000.
2. R. Rajkumar, Synchronization in Real Time Systems: Priority Inhertiance Approach, Kluwer Academic Publishers, 1991.
3. J. Stankovic, M. Spuri, K. Ramamritham, G. C. Butazzo, Deadline Scheduling for Real Time Systems: EDF and Related Algorithms, Kluwer Academic Publishers, 1998.
4. J. W. S. Liu, Real-Time Systems, Prentice Hall, 1998.
5. B. Lewis, D. J. Berg, Multithreaded Programming with Java Technology, Prentice Hall, 2000.
6. D. Lea, Concurrent Programming in Java, Prentice Hall, 2000.
7. H. M. Dietel, Operating Systems, Addison-Wesley, 1990.
8. T. Lindholm, F. Yellin, The Java Virtual Machine Specification, Second Edition, Addison-Wesley, 1999.
9. J. Gosling, B. Joy, G. Steele, The Java Language Specification, Addison-Wesley, 2000.
10. B. Venners, Inside the Java Virtual Machine, McGraw-Hill, 1999.
11. E. Giguere, Java 2 Microedition: A Professional Developer's Guide, John Wiley & Sons, 2000.
12. G. Bollella, J. Gosling, D. Hardin, The Real-Time Specification for Java, Addison-Wesley, 2000.
13. E. Bertolissi, C. Preece, Java in Real-time Application, IEEE Trans. on Nuclear Science, Vol. 45, 1996.
14. W. Foote, Real-time Extenisions to the Java Platform A Progress Report, Sun Microsystems.
15. K. Nilsen, Issues in the Design and Implementation of Real-Time Java.
16. A. Di Stefano, C. Santoro, A Java Kernel for Embedded Systems in Distributed Process Control, IEEE Concurrency, October-December, 2000.
17. K. Nilsen, Invited Note: Java for Real-Time.
18. G. Bernat, A. Burns, A. Wellings, Portable Worst-Case Execution Time Analysis Using Java Byte Code.
19. E. D. Jensen, A Proposed Initial Approach to Distributed Real-Time Java, The MITRE Corporation.
20. C. Lizzi, Enabling Deadline Scheduling for Java Real-Time Computing.
21. G- Hilderink, A. Bakkers, J. Broenink, A Distributed Real-Time Java System Based on CSP.
22. U. Brinkschulte, C. Krakowski, J. Kreuzinger, T. Ungerer, A Multithreaded Java Microcontroller for Thread-Oriented Event-Handling.
23. G. Xydas', J. Tassel, Experimentation in CPU control with Real-Time Java.
24. G. A. Agha, Concurrent Java, IEEE Concurrency, October-December 1997.