CIS 721

Real Time System

Discovering key features of Real Time Java

Report

Karl Remarais

Spring 2011

1. Introduction

The objective of this report is to present the key features of java real time. Our survey will first describe the obstacles that prevent the use of standard Java for the implementation of real time software. Second, we describe how java real time offers a viable solution to many of the weaknesses of java standard. Finally, we will present a simple demo program that calls upon some of the features that are introduced by the Real Time Java Specification (RTJS).

2.- Issues with java standard

Many characteristics of java standard make it difficult to implement real time software. Real time system requires both logical correctness and temporal correctness. The same input under the same circumstances must produce the same result. Constraints such as execution time, deadline, and period must be followed to avoid potentially fatal risks. Here after we present obstacles that prevent implementing real-time program in standard java:

First, threads management in java standard does not propose an effective real-time scheduling policy. There is no strict priority control over threads and this is essential to build real-time applications. Second, the just-in-time compilation (JIT), which is supposed to improve run time performance, may introduce some latency in the execution of a task because the JIT compilation may occurs at any moment. Third, the garbage collector is another source of uncertain latency as it has a non-deterministic behavior.

Second, standard java has some limitations over time resolution. Although java SE proposes two classes Date and Calendar, the time resolution is in the order of 1ms, which is not precise enough for many real-time applications. Moreover, timers do not meet deadlines. They wait at least the specified delay.

Finally, java SE does not provide means to access the physical memory. Although access to the physical memory is not always required for real time applications, it is useful in many situations. Because of these “weaknesses” java standard is unsuitable for implementing real time system.

3.-RTSJ provides enhancements

In this section we present the solutions java real time proposes to the deficiency previously mentioned. Java RTS introduces a new package of javax.realtime containing 3 interfaces, 47 classes and 13 exceptions. It is totally compatible to classical java application in term of syntax and also any java SE software should be able to run on a JVM RTSJ. It introduces two new types of threads: RealtimeThread and NoHeapRealtime, asynchronous event handler, fine time management with AbsoluteTime and Realtime classes, and finally gives access to physical memory. Below we investigate succinctly the enhancements of java RST.

The first enhancement of RTSJ is the predictability of a program in execution in term of threads management. The RTSJ specifies a PrioriotySchedurer (fixed priority and preemptive) as the minimum scheduler in any implementation of the RTSJ. It also requires a minimum of 28 real-time priority levels that are higher to the regular real-time threads. FIFO algorithm is applied within a set of threads with the same priority. Moreover, the JVM changes threads priority only when there is a priority inversion situation. With the default priority scheduler the RTSJ provides the means to build other scheduler such as RMS or EDF. The scheduler relies on two classes SchedulingParameters and ReleaseParameters that are associated to real time threads and guide the decisions of the scheduler.

> The SchedulingParameters is an abstract class. In our demo program (section 4) we use one of its subclasses, PriorityParameters, which the RTSJ defines for the required scheduler (PriorityScheduler). It holds the priority of the real time thread. However, in a more complex scheduling policy, the idea is to subclassed the ShedulingParameters to add detail parameters that will used by the scheduler to determine eligibility of tasks.
>
> The ReleaseParameters contains information about threads released information. A thread may be released periodically, sporadically or aperiodially. The ReleasedParameters also encapsulate the start time, cost per released, deadline and handlers for missed deadlines or cost overruns. In our demo program we use PeriodicParameters which is a subsclass of ReleaseParameters. ReleaseParameters may be used by many different RealtimeThread it is a one-to-many relation. Once the parameters of the ReleaseParameters changes it changes for all the threads that are associated to it.

Second, the RTSJ provides some mechanism to handle synchronization and avoid priority inversion. It requires that at least one priority inversion avoidance protocols to be implemented. The default one is the priority inheritance protocol that ensures a low priority thread may temporarily inherits the priority of the higher priority thread that is waiting. In addition, the RTSJ specifies that any implementation of java real-time specification should implement wait free queues to provide protected, non-blocking, shared access to objects accessed by both regular Java threads and NoHeapRealtimeThreads.

Third, the RTSJ introduces tree categories of memory. The heap is managed by the garbage collector. The immortal memory or "permanent memory" is not managed by the garbage collector but is freed at the end of the application. Finally, the scope memory is not subject to the garbage collector, but instead has the life time of the object that uses them. In addition the RTSJ provides mechanism to access directly the physical memory. Physical memory can be of type PhysicalMemory Immortal or Scoped. It serves to see specific addresses of input output. It must contain only string of bytes (RawMemoryAccess). In our demo program the real time threads run in LTmemory which extends javax.realtime.ScopedMemory.

Finally, the RTSJ introduces various kinds of asynchronous events and event handling. When an event occurs the associated handler is executed. For instance, in our implementation each deadline miss fires an event that is handled by the miss deadline class. The RTSJ also proposes some other asynchronous mechanism such asynchronous thread termination an asynchronous transfer of control.

3.1 Performance of RTSJ (RT JVM)

Various performance evaluation of real time java has demonstrated that the RTSJ is very efficient. For instance, result of tests conducted for the Boeing Company with JTime RT JVM, which is a commercial implementation of the RTSJ, showed that RTSJ is capable of meeting the requirements for mission critical embedded system performance [2]. However, the same test conducted with a C++ version of the program used for the evaluation reveals that java real time is slower. The author attributes the difference to byte code interpretation. In another paper [1] the authors evaluated an open source implementation of RTSJ (jRate) along with the JTime RT JVM

implementation of TimeSys. The result of the test suggests similar conclusion about the efficiency of RTSJ and the maturity of real time java. Java Real Time can be used to implement real time system and embedded real-time system.

3.2 Implementations of the Real-Time Specification for Java (RTSJ)

Currently there are various implementation of the RTSP. In the previous section we mentioned JTime the RI JVM which is a commercial implementation developed by TimeSys. Among other implementation are the IBM's WebSphere Real Time, Sun Microsystems's Java SE Real-Time Systems, Aonix PERC and JamaicaVM from aicas.

3.4 Future of java RTSJ

The future of java real time seems promising. First and foremost, the number of java developers is significant and the compatibility with java SE makes the development of real time program “easier”. As real-time embedded computing is becoming more in more ubiquitous there is a need for cheaper cost for developing real time embedded software. Java real time could be best solution to satisfy the increasing demand of soft real time software.

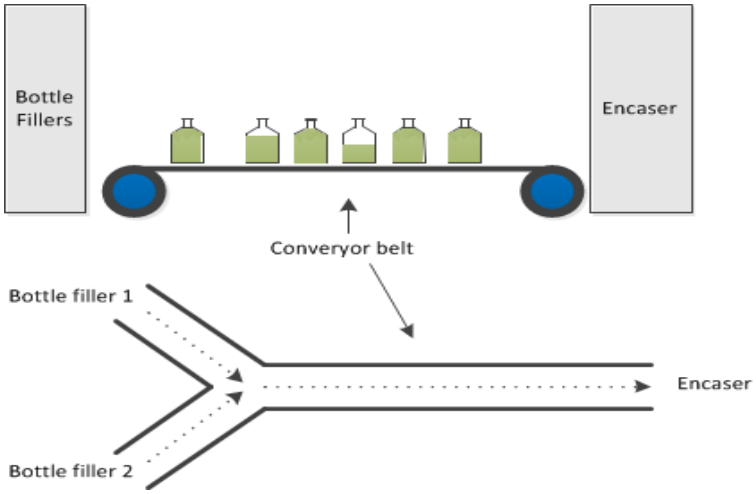
4. Demo

This section describes the implementation of a demo application using real time java. The goal is to try via a practical introduction some of the features of java RTS.

4.1 Scenario description

For this implementation we consider the following scenario. In a brewery two bottle fillers are putting bottles on a conveyor belt. An encaser at the other extremity of the conveyor belt is removing them while bottles are approaching. The bottle fillers and the encaser are modeled as real-time threads. Both classes BottleFiller and Encaser are subsclasses of the javax.realtime.RealtimeThread. The class ConveryorBelt models the conveyor as array of int the

size of the array is given as parameter. The ConveyorBelt includes two synchronized methods add and remove that respectively simulates the action of adding and removing a bottle from the conveyor.



The classes MissHandler and OverrunHandler respectively handle the asynchronous event deadline misses and cost overrun. The structure of the code is the following:

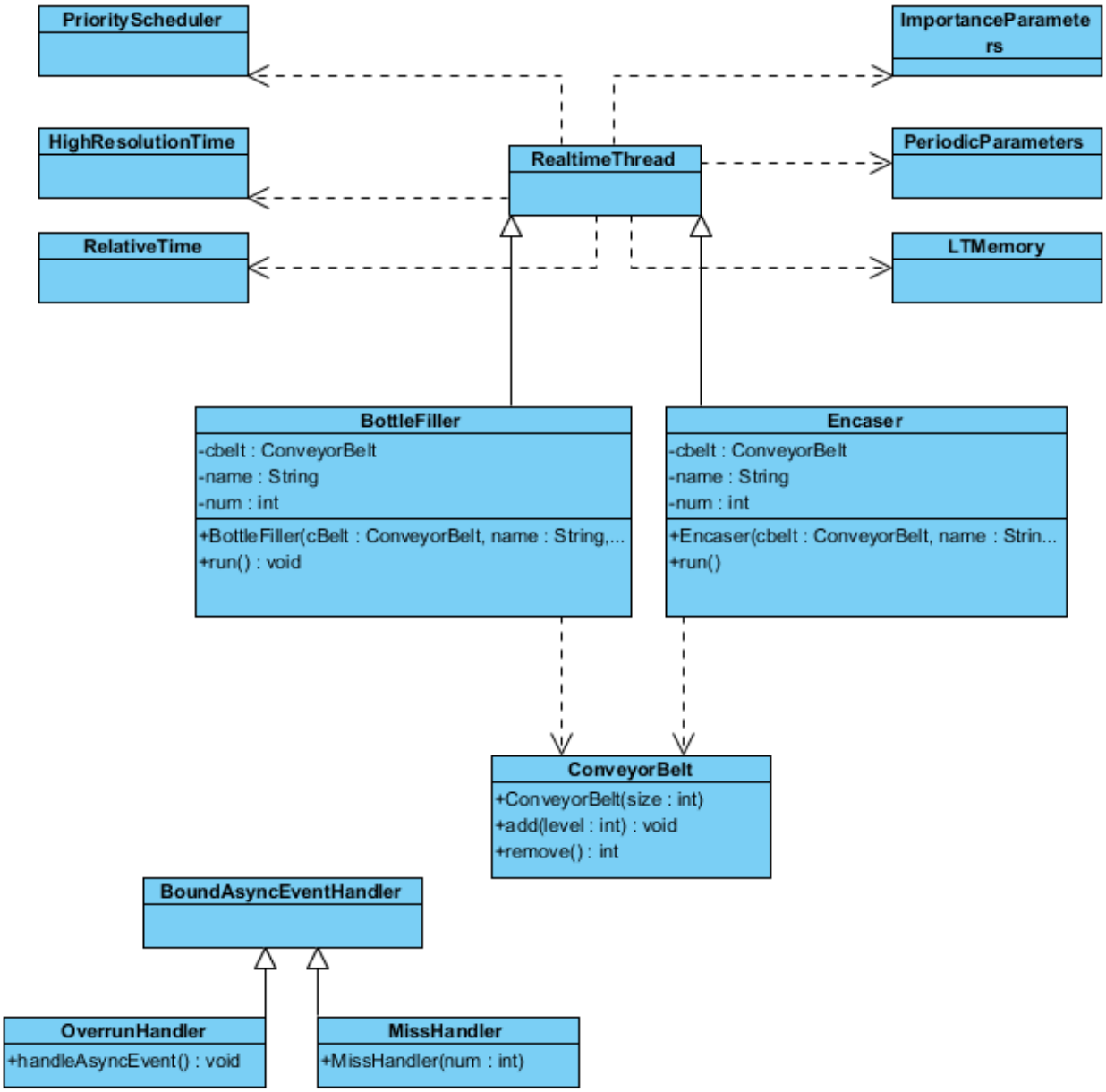
In the main method of the Class Main we first create an object PrioritySheduler. Using this default scheduler we assign priorities to our real-time threads. The highest priority is assign to the encaser thread since it has shortest period. Similarly, priorities are assigned to the two other threads in decreasing order of priority.

```
PriorityScheduler sched = (PriorityScheduler) Scheduler.getDefaultScheduler();
TBasePriority = sched.getMaxPriority();

TPriority[0] = (sched.getMaxPriority()) - 1;
TPriority[1] = (sched.getMaxPriority()) - 2;
TPriority[2] = (sched.getMaxPriority()) - 3;
```

After the creating and starting the real-time threads the main thread enter a loop whose exit condition is all real-time threads are done.

For this demo we use the following environment: Ubuntu 10.10 and Sun Java RTS Evaluation Program. The demo program is composed of 7 classes. *(For the missing libcap.so.1 problem the solution is to locate the libcap.so.2 and rename it as libcap.so.1)*



4.6 Evaluation

In our implementation the thread are consuming the LTmemory that is defined for executing the program. We identified the source of the problem is because the various System.out.println() used in the program to send status to the console is consuming part of the memory each time it is called. In addition, because the garbage collector is not allowed to free objets in scope memory

until the end of the execution of the thread, the successive calls tends to consume progressively the initial memory that was allowed to the realtimethread. Solution to solve the problem is to define a specific chunk of memory to run the System.out.println() in that same chunk whenever it is necessary.

5. Conclusions and Future Work

This work provides us an effective introduction to the RTSJ. In the future we shall investigate more about how to handle gracefully communication between different types of threads for instance, common threads and the RealtimeThread or NoHeapRealtimeThread. We are already aware that the RTSJ provides wait-free synchronization: the queue classes WaitFreeWriteQueue and WaitFreeReadQueue, to enable communication between regular threads and real-time threads. Second, we shall also try to examine closely the mechanism of nested entry and exit in to scope memories and the lifetimes of object allocated in these scope memories.

References

[1] E. Bruno, G. Bogella. "Real-time java Programming with Java RTS"

[2] A. Corsaro and D. C. Schmidt, "Evaluating Real-Time Java Features and Performance for Real-time Embedded Systems".

[3] http://jrate.sourceforge.net/

[4] Sun Java Real-Time System http://java.sun.com/javase/technologies/realtime/index.jsp

[5] T. Henties et al. (2009) "Java for Safety-Critical Applications"

[6] P. Tsigas, Y Zhang, D. Cederman, T. Dellsen. "Wait-free Queue Algorithms for the Real-time Java Specification"

Output

```
System default parameters.
for bottle filler1 realtime thread:  period==3000; cost=1000; dealine= 1500
for bottle filler2: realtime thread  period==4000; cost=1000; dealine= 1500
for encaser : realtime thread:  period==5000; cost=1000; dealine= 1500

Name      Priority      Activation                Period          Cost
Deadline
BF-0 : 57      (1305348875456 ms, 598901 ns)(3000 ms, 0 ns)(1000 ms, 0 ns)(1500 ms, 0
ns)
BF-1 : 56      (1305348875456 ms, 598901 ns)(4000 ms, 0 ns)(1000 ms, 0 ns)(1500 ms, 0
ns)
E- 1 : 55      (1305348875456 ms, 598901 ns)(5000 ms, 0 ns)(1000 ms, 0 ns)(1500 ms, 0
ns)
Starting time of threads : (1305348874468 ms, 424682 ns)
  activation date  : (1305348875456 ms, 598901 ns)
Start: BF-0
Start: BF-1
Start: E- 1
BF-0  start :(1305348875457 ms, 940784 ns)
BF-0new bottle on belt :
BF-1  start :(1305348875458 ms, 152328 ns)
BF-1new bottle on belt :
E- 1  start :(1305348875458 ms, 278683 ns)
E- 1one bottle remove :
BF-0new bottle on belt :
BF-1new bottle on belt :
E- 1one bottle remove :
BF-0new bottle on belt :
BF-1new bottle on belt :
BF-0new bottle on belt :
E- 1one bottle remove :
BF-0new bottle on belt :
BF-1new bottle on belt :
BF-0new bottle on belt :
E- 1one bottle remove :
BF-1new bottle on belt :
BF-0new bottle on belt :
BF-1new bottle on belt :
E- 1one bottle remove :
BF-0new bottle on belt :
BF-0new bottle on belt :
```

```
BF-1new bottle on belt :
E- 1one bottle remove :
BF-0new bottle on belt :
BF-1new bottle on belt :
BF-0new bottle on belt :
E- 1one bottle remove :
BF-1new bottle on belt :
BF-0new bottle on belt :
E- 1one bottle remove :
BF-0new bottle on belt :
BF-1new bottle on belt :
BF-0new bottle on belt :
BF-1new bottle on belt :
E- 1one bottle remove :
BF-0new bottle on belt :
BF-1new bottle on belt :
BF-0new bottle on belt :
E- 1one bottle remove :
BF-0new bottle on belt :
BF-1new bottle on belt :
E- 1one bottle remove :
BF-0new bottle on belt :
Thread[RealtimeServerThread-3,26,] : Miss deadline :BF-1
BF-0new bottle on belt :
E- 1one bottle remove :
Thread[RealtimeServerThread-3,26,] : Miss deadline :BF-1
Thread[RealtimeServerThread-1,26,] : Miss deadline :BF-0
--- main : BF-0 alive
BF-0new bottle on belt :
E- 1one bottle remove :
Thread[RealtimeServerThread-3,26,] : Miss deadline :BF-1
Thread[RealtimeServerThread-1,26,] : Miss deadline :BF-0
Thread[RealtimeServerThread-1,26,] : Miss deadline :BF-0
BF-0new bottle on belt :
E- 1one bottle remove :
Thread[RealtimeServerThread-3,26,] : Miss deadline :BF-1
Thread[RealtimeServerThread-1,26,] : Miss deadline :BF-0
Thread[RealtimeServerThread-3,26,] : Miss deadline :BF-1
BF-0new bottle on belt :
E- 1one bottle remove :
Thread[RealtimeServerThread-1,26,] : Miss deadline :BF-0
Thread[RealtimeServerThread-3,26,] : Miss deadline :BF-1
```

```
Too many many deadline miss stop conveyor belt and bottle filler! Count =11
Thread[RealtimeServerThread-1,26,] : Miss deadline :BF-0
Too many many deadline miss stop conveyor belt and bottle filler! Count =12
Java Result: 1
BUILD SUCCESSFUL (total time: 1 minute 15 seconds)
```