Java e Sistemas de Tempo Real: Vale a pena utilizar?

Sabemos que uma das prerrogativas da linguagem de programação Java "Write once, run anywhere!", escreva uma vez e execute em qualquer plataforma, em outras palavras portabilidade. Tal portabilidade é permitida principalmente pela JVM (*Java Virtual Machine*), pois o programa é compilado primeiro em JBC (*Java byte code*), para então ser executado pela JVM.

Em sistemas de tempo real são esperados que os mesmos produzam resultados em estritas quantidades de tempo. Portanto, é crucial conhecer o tempo de execução do pior caso (*WCET – Worse Case Execution Time*) e garantir que ele cumpra a sua "deadline".

Características da linguagem Java e suas tradicionais JVMs, como o uso de múltiplas *threads*, compilação dinâmica, *garbage collector* (GC) automático, entre outras, resultam em muita imprevisibilidade. Entretanto desde os anos 1990 discussões sobre implementações de Java para tempo real, culminaram no desenvolvimento da *Real-Time Specification for Java* (RTSJ) que foi aprovada em 16 de dezembro de 1998 (JSR 1), assim como a *Safety Critical Java™ Technology* (JSR 302) em 2006. Entre as implementações comerciais de Java para sistemas de tempo real podemos citar *IBM WebSphere Real Time*, *TimeSys jTime* e *Aicas JamaicaVM*.

Além das *threads* normais de Java, outros dois tipos são definidos pela RSTJ: *real-time thread* (RT) e *no-heap real-time thread* (NHRT). NHRTs não podem acessar a pilha de memória, evitando o atraso e a imprevisibilidade causados pelo *garbage collector*. Entretanto, as RTs podem acessar a pilha de memória, permitindo uma maior flexibilidade, adequados para aplicações *soft real-time*, que possuem uma maior tolerância relacionado à atrasos.

Uma das principais formas de evitar a imprevisibilidade do *garbage collector*, é contornar o uso do mesmo. Para tal, estão definidos na RTSJ o uso de *scoped memory* (memória com escopo) e *immortal memory* (memória eterna), para o gerenciamento de memória nas implementações em tempo real. Usando memória eterna, recursos são compartilhados entre todas as *threads*, e diferentemente objetos normais em Java, eles continuam a existir na pilha mesmo que não haja referências a eles. Já na memória com escopo, os objetos existem na memória tanto quanto existam *real-time threads* que os acessem. Várias melhorias tem sido feitas e estudadas com relação a essa forma, assim como outras formas estão sendo estudadas e melhoradas para sanar o problema da imprevisibilidade relacionada ao GC.

Conclusão

Devido tanto à demanda atual de sistemas embarcados e sistemas de tempo real, quanto à robustez, flexibilidade e por ser amplamente difundida entre desenvolvedores, acredito que "vale a pena utilizar" Java para a implementação de sistemas de tempo real, tanto para sistemas *hard real-time* quanto para *soft real-time*, principalmente. Embora implementações relacionadas à sistemas *hard real-time* precisem do uso de classes como *ManagedSchedulable* (definido na JSR 302) e *NoHeapRealTimeThread* (definido na JSR 1), garantindo a previsibilidade do sistema, porém perdendo a sua flexibilidade no desenvolvimento do mesmo.

Referências

J. Gosling, G. Bollella. The Real-Time Specification for Java, Boston, MA: Addison-Wesley, 2000.

J. Kwon, A. Wellings, and S. King. “Ravenscar-Java: a high integrity profile for real-time Java” in *Proceedings of the 2002 Joint ACM-ISCOPE Conference on Java Grande*, Seattle, WA, 2002, pp. 131-140.

JSR 1: Real-time Specification for Java. Disponível em: < https://jcp.org/en/jsr/detail?id=302>. Acessado em 29 set 2015.

JSR 302: Safety Critical JavaTM Technology. Disponível em: < https://jcp.org/en/jsr/detail?id=302>. Acessado em 29 set 2015.

K. Nilsen. Developing Real-Time Software with Java SE APIs: Part 1. 2014. Disponível em: <http://www.oracle.com/technetwork/articles/java/nilsen-realtime-pt1-2264405.html>. Acessado em 29 set 2015.

W. Zhang, Y. Sun. Overview of Real-Time Java Computing. 2013. Disponível em: <http://jcse.kiise.org/files/V7N2-02.pdf >. Acessado em 29 set 2015.