Perché Java non è una piattaforma valida per i sistemi real - time.

La piattaforma Java e le sue estensioni sono estremamente diffuse, tanto da raggiungere ambiti molto diversificati, dalla gestione di transazione sicure a quella di flussi multimediali, senza dimenticare le piattaforme mobili ed i sistemi embedded.

La diffusione di Java nei sistemi real time è stata fortemente limitata dal fatto che i tempi di esecuzione di un'applicazione Java sono difficilmente predicibili.

Anche qualora la Java Virtual Machine si appoggi su un sistema operativo adatto ai sistemi in tempo reale la predicibilità dei tempi di esecuzione di un programma Java è fortemente limitata da fattori interni alla Virtual machine. Questi fattori sono principalmente collegabili alle attività interne alla Virtual Machine.Queste ativitàpossono causare la sospensione temporanea di un processo attivo. In questa sezione si presentano le principali cause che causano significative variazioni nei tempi di esecuzione per la stessa applicazione (jitter).

Lazy initialization

La Java VM normalmente inizializza una classe quando il programma tenta per la prima volta di ottenere un'istanza di essa, o quando tenta di utilizzare un metodo statico o una variabile statica della classe stessa. Quando la classe è inizializzata, le sue variabili statiche sono inizializzate. Questa attività, ovviamente, richiede tempi di esecuzione che possono interferire con l'esecuzione di codice critico per l'applicazione in tempo reale. Poiché non è possibile prevedere quando le classi saranno inizializzate, non è possibile accertarsi di quando questo accadrà, né se la quantità di ritardo introdotto sarà accettabile.

Just In Time Compilation

Just in Time (JIT) è la modalità di compilazione standard di una macchina virtuale Java. La compilazione JIT seleziona i metodi che devono essere compilate sulla base della frequenza della loro esecuzione. La compilazione di un metodo viene attivata in due situazioni:

Quando una chiamata interpretata viene eseguita, una delle prime operazioni effettuate dalla macchina virtuale è quella di incrementare il contatore di invocazione del metodo chiamato. Se il valore del contatore raggiunge la soglia di compilazione, il metodo viene compilato prima di essere eseguito.

Il numero di iterazioni di eventuali cicli all'interno del metodo è monitorata da un contatore. La compilazione viene attivato anche quando questo contatore raggiunge un valore soglia. Questo processo è noto anche come On – Stack Replacement(OSR).

Il momento in cui avviene la compilazione è, quindi, totalmente sotto il controllo della macchina virtuale e non può essere previsto nella totalità dei casi. Per esempio, se una sezione di codice critica viene eseguita raramente, il contatore può raggiungere la soglia che fa scattare la compilazione dopo un tempo molto lungo.

Tuttavia non c'è quindi modo di sapere quando la compilazione si verificherà e questa incertezza non è accettabile in una applicazione real-time.

Tramite un'opzione si può istruire la Virtual Machine affinchè esegua la compilazione in background. In questa modalità, il thread non viene interrotto per permettere la compilazione del metodo che sta per eseguire: il thread esegue il metodo in modalità interpretata e la compilazione viene differita. Bisogna tuttavia sottolineare come, se il carico sulla cpu è elevato, la compilazione rischia di essere differita per un tempo indefinito, facendo sì che l'applicazione venga sempre eseguita in modalità interpretata, con grandissima inefficienza.

Garbage Collection

Il servizio di Garbage Collection all'interno della Virtual Machine consente di recuperare aree di memoria precedentemente occupate, ma non più accedibili e di renderle quindi utilizzabili per l'allocazione di nuovi oggetti. La strategia per localizzare queste aree di memoria da considerare come riutilizzabili all'interno del java heap è, appunto, quella di individuare gli oggetti che non sono più accedibili tramite da nessuno dei thread attivi all'interno della macchina virtuale. Per fare ciò si possono adottare varie metodologie, la maggior parte delle quali ha bisogno di lavorare su un'istantanea del sistema. Per questo motivo, nella Java Virtual Machine standard tutte le esecuzioni vengono fermate quando entra in azione il garbage collector(stop the word).

Appare evidente come l'attivita di garbage collection, che prende il via ogni qual volta la memoria libera nell'heap scende sotto una determinata soglia, sia una forte fonte di indeterminismo. E' infatti difficile prevedere quando il garbage collector entrerà in esecuzione, dal momento ciò è legato quota di memoria libera nell'heap, valore dell'andamento difficile da prevedere, specie considerando la pluralità di applicazioni che spesso sono in esecuzione sulla stessa Virtual Machine. Altro fattore di difficile stima è il tempo impiegato dal garbage collector per adempiere al suo compito: questo è infatti fortemente influenzato da fattori quali la quantità di memoria da riciclare e ad altri fattori legati ai singoli algoritmi di garbage collecting, quali, ad esempio, la profondità e il numero di connessioni dei percorsi che partono dagli oggetti radice o al numero di oggetti presenti nel sistema.

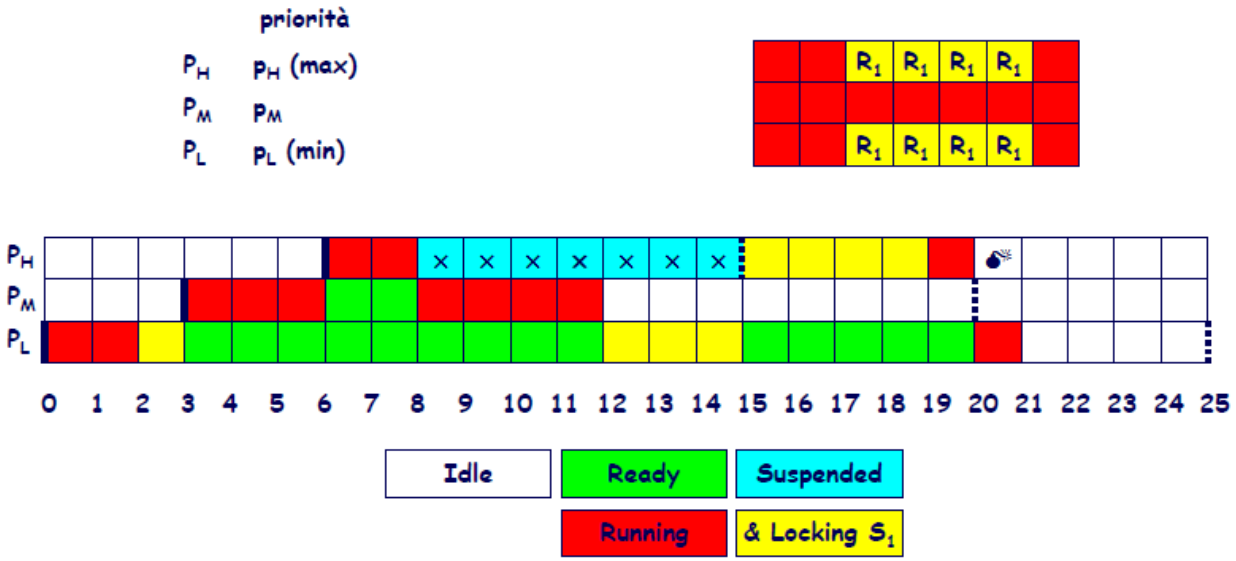
Scheduling

Java mette a disposizione dello sviluppatore alcuni metodi della classe Thread che consentono di influenzare il comportamento dello scheduler interno alla Virtual Machine. Sono presenti dieci livelli di priorità da associare ai thread. Inoltre, tramite il metodo yeld è possibile posizionare il thread in fondo alla coda dei processi pronti relativa alla sua priorità.

Tuttavia, nell'ottica di sviluppo di applicazioni in tempo reale, questi strumenti non sono soddisfacenti. In particolare non è garantito che, tra i processi pronti, sia sempre in esecuzione quello a priorità maggiore. Questo rende difficile stabilire quanto tempo un processo impiega per la sua esecuzione, dal momento che un thread può risultare bloccato anche a causa di un thread con priorità inferiore. Inoltre la virtual machine è libera di usare politiche di time-slicing per processi che eseguono alla stessa priorità. Questo fa sì che il tempo di esecuzione di un thread possa venire artificiosamente dilatato a causa della frammentazione dell'esecuzione stessa in più slice intervallati tra di loro.

Sincronismo

Gli strumenti che Java mette a disposizione per sincronizzare i Thread sono sostanzialmente basati su lock a mutua esclusione. Seppur dalla comprovata validità nel caso generale, queste soluzioni non sono da sole soddisfacenti per i sistemi in tempo reale, in quanto prone a fenomeni indesiderati quali l'inversione incontrollata di priorità di cui si mostra un esempio in figura.



Nell' esempio il processo a priorità maggiore quando tenta di accedere alla risorsa R1 si sospende perché questa è già occupata dal processo Pl. Tuttavia non è accettabile che il processo Pm eserciti indirettamente preemption su un processo a priorità maggiore come Ph, pur non condividendo con esso nessuna risorsa. Per evitare questo genere di fenomeni sono stati sviluppati in letteratura una serie di protocolli (tra i più noti si citano priority inheritance e priority celing) che la versione standard di Java non supporta.