JAVA REAL-TIME: ANALISI DELL'ARCHITETTURA E VALUTAZIONE SPERIMENTALE DELLA PIATTAFORMA SOLARIS 10.9

Candidato:
Marco Nanni

RELATORE:
Chiar.mo Prof. Eugenio Faldella

CORRELATORI
Chiar.mo Prof. Marco Prandini
Dott. Ing. Primiano Tucci

sommario

- I sistemi in tempo reale
- I limiti di Java come piattaforma in tempo reale
- La specifica Java Real – Time (RTSJ) ed il sistema RTJS su SO solaris 10.9 – caratteristiche peculiari
- I moduli di busyWait e di Logging
- Lo scheduling di default e lo scheduler EDF
- Analisi della politica di default di gestione dei deadline miss e delle sue criticità
- Realizzazione della politica SKIP in java Real-Time
- Realizzazione della politica SKIPSTOP in java Real-Time

I sistemi in tempo reale

- Caratteristiche dei sistemi in tempo reale
  - Necessità di fornire una risposta entro un tempo massimo
  - Presenza di processi periodici
- Esempi
  - Aereo
  - Missili intercettori
  - Plant
  - Diffusione in altri campi: automotive, automazione ind, multimedia
- La crescente diffusione e complessità dei sistemi in tempo reale richiede l'introduzione di linguaggi già strutturati e con una curva di apprendimento poco ripida

Java ed i sistemi in tempo reale

- Java piattaforma diffusa, ma con caratteristiche che ne limitano l'uso nei sistemi in tempo reale
  - Lazy initialization
  - JIT complitation
  - Garbage collecion
  - Sistema soggetto ad inversioni incontrollate di priorità

La specifica Java real-time

- La specifica Java real-time, la cui ultima versione risale al 2006, propone una serie di estensioni e modifiche a java standard al fine di rendere java un linguaggio adatto ai sistemi in tempo reale
- La specifica prevede:
  - Nuove classi per misurare e comparare il tempo
  - Nuove zone di memoria dove allocare gli oggetti
  - Meccanismi di gestione di eventi asincroni e di segnali POSIX
  - Politiche (priority inheritance e priority ceiling per evitare fenomeni di inversione incontrollata di priorità)
  - Meccanismi di scheduling più raffinati
- Package javax.realtime

Dentro la specifica: gli oggetti schedulabili

- Due tipi di oggetti attivi RealtimeThread e AsyncEventHandler (implementano l'interfaccia Schedulable)
- Ogni oggetto attivo ha associato una serie di parametri che ne caratterizzano l'esecuzione
  - (al posto dell'elenco metti una bella figura)
  - ReleaseParameters
  - SchedulingParameters
  - MemoryParameters
  - Processing group parameters

Realtime Thread

- Estende java.lang.thread
- Esibisce tre metodi statici utili in caso di processo periodico
  - waitForNextPeriod
  - Deschedule Periodic
  - Schedule Periodic

AsyncEventHandler

- Risponde alla necessità di reagire ad eventi asinroni(come una violazione di deadline)
- Non entra immediatamente in esecuzione (dipende dai parametri di scheduling)
- Quando si verifica un evento
  - Un thread realtime di sistema esegue il suo metodo handleAsyncEvent
  - Se binding dinamico troppo costoso -> BoundAsyncEventHandler

Il sistema oggetto della tesi

- Real-Time Java System (RTJS), sviluppato da Sun-Oracle.
- Versione 2.2 academic per Solaris 10.9
- Caratteristiche peculiari
  - Liste di preinizializzazione e compilazione
  - Real Time Garbage Collector
    - Non ercita preemption su thread real time
    - Può eseguire concorrentemente (assenza di fasi stop the world)

Prime estensioni: BusyWait

- Necessità di modellare l'esecuzione di lunghezza desiderata
- Non si può usare sospensione perché il thread può subire preeemption ed il tempo va comunque avanti
- Occorre occupare la cpu per un tempo desiderato con operazione di nessuna utilità
- Il componente deve essere riutilizzabile su sistemi con capacità computazionali diverse
- Prima di poter essere utilizzato va inizializzato in modo che calcoli un valore che esprime quante iterazioni il sistema è in grado di eseguire in certo lasso di tempo.
- Dualmente, si ricava il numero di iterazioni necessarie per eseguire una busyWait di durata desiderata

Prime estensioni: logging

- Necessità di monitorare l'attività delle varie entità durate l'esecuzione, senza incorrere in rallentamenti e problemi di sincronizzazione
- Una zona di memoria riservata per ogni entità
- Le scritture sono il più rapide e sintetiche possibile
- A fine dell'esecuzione una serie di utility permettono di unire i fari log e fornire un risultato in forma user- friendly

Lo scheduling in Java Real - Time

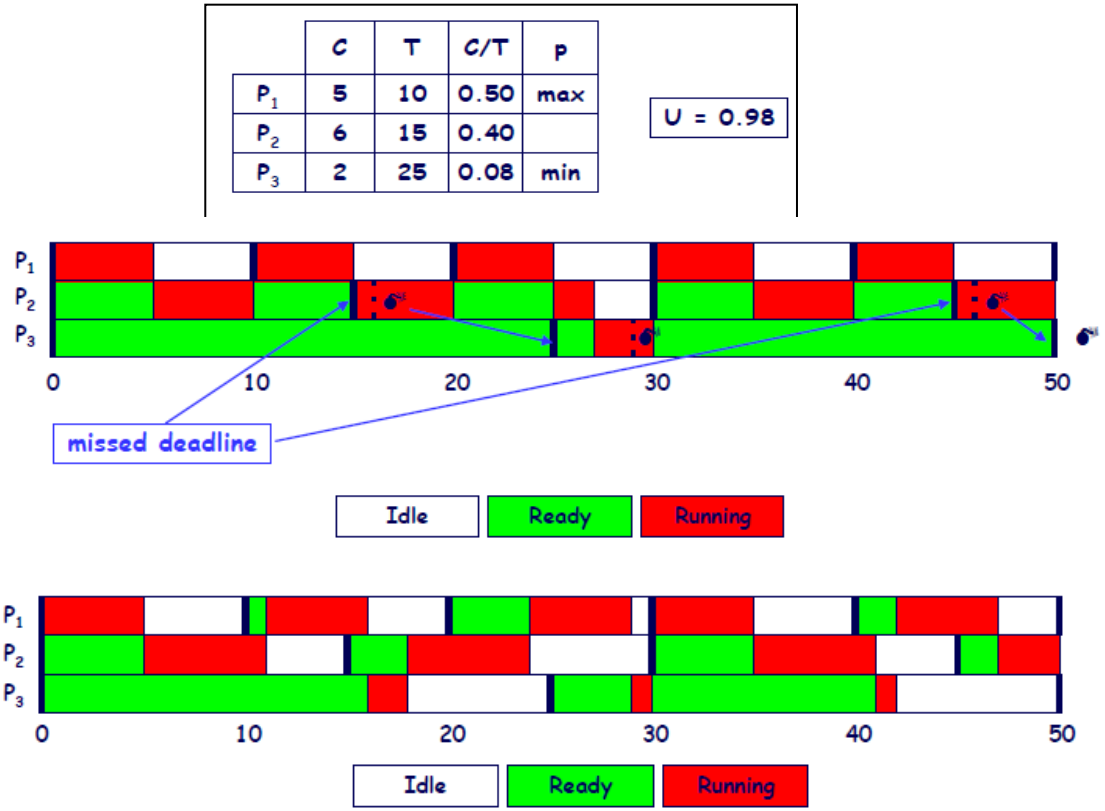
- La specifica prevede solo uno scheduler basato su priorità statica (PriorityScheduler)
- Entra in esecuzione il thread a priorità maggiore (priorità espressa dall'utente nei PriorityParametres)
- RTJS implementa questa specifica mappando direttamente i thread Java nei thread di sistema.
  - Di fatto, è il sistema operativo che provvede a mettere in esecuzione il thread più prioritario

Uno scheduler EDF per java Real-Time

- Lo scheduler pone in esecuzione il thread che ha la deadline più imminente. Lo scheduler è pensato per un sistema monoprocessore, sepur è stato pensato per esser facilmente esteso al caso multiprocessore
- Tiene i processi pronti in una coda a bassa priorità ordinati per deadline
- Quattro livelli di priorità
  1. Scheduler priority
  2. Handler priority
  3. excecutingPriority
  4. Ready Priority
- I processi eseguono un prologo ed un epilogo a priorità massima
  - Il prologo calcola la prossima dedline ed inserisce il thread nella coda dei processi pronti o lo pone in esecuzione
  - L'epilogo pone il thread a priorità massima perché non subisca preemption al prossimo avvio e mette in esecuzione il primo thread della coda
- Grazie a priority inherintance scheduler robusto anche in caso di blocco su accesso a risorse condivise da parte del processo in esecuzione

Confronto performance tra EDF e RMPO

- Qui, al posto dei grafici delle slide metti due belle finestre di TSV



Gestione dei deadline miss

- I PeriodicParameters permettono di caratterizzare un thread periodico. Questi parametri permettono di specificare
  - Periodo
  - Deadline
  - Gestore di deadline miss (un AsyncEventHandler che viene richiamato se il thread non ha terminato il job prima della deadline)

La politica di default di Java real-time in caso di deadline miss

- Se non si specifica un handler, chiamata a waitForNextPeriod non bloccante.
- Con un handler specificato necessità di chiamare il metodo schedulePeriodic sul thread
- In ogni caso l'esecuzione del job non viene interrotta
- Il comportamento è conforme alla politica ASAP

Limiti della politica ASAP

- Qui metti grafico per illustrare comportamento

- Non aiuta ad alleggerire situazioni di sovraccarico del sistema
- L’esecuzione del job può durare una quantità indefinita di tempo bloccando il sistema

L’implementazione della politica Skip

- La poltica skip prevede di non schedulare nessun altro job nel periodo in cui termina l’esecuzione del job che ha violato la deadline
- Figura

- Il sistema, invece, schedulerebbe tanti job quante le deadline mancate
- Si implementata questa politica per un sistema monoprocessore, con thread puramente periodici

L’implementazione della politica Skip

- Metti figura gestore e periodicThread
- PeriodicThread – Modella thread periodico
- DeadlineMissHandler – Modella handler di deadline miss
- IPendingJobManager – rappresenta il gestore dei job “di recupero”.
  - Implementato dall’handler in modo che tutta la politica sia contenuto in esso
- Flag pending mode permette al thread di distinguere tra un job normale ed uno di recupero

L’implementazione della politica Skip

- Qui metti diagramma di interazione
- Quando si verifica deadline miss handleasyncevent mette flag a true e incrementa skipCount
- Al job successivo periodicThread chiama dopendigJob che decrementa il contatore. Se questo ha raggiunto il valore zero si reimposta il flag a false

risultati

- Metti grafico comparato con asap per fare vedere che saltando i job non c’è effetto domino

Il trasferimento asincrono di controllo

- IPer evitare che un job potesse eseguire per un periodo indefinito di tempo si è creato handler che , sfruttando il trasferimento asincrono di controllo, provvedesse ad interromperlo se viola consecutivamente un certo numero di deadline
- Il trasferimento asincrono di controllo permette di definire un metodo interrompibile tramite l'inserimento di una AsynctronouslyInterruptedException nella sua throw list.
- Se si chiama il metodo interrupt mentre un thread real-time sta eseguendo un metodo inettompibile viene sollevata una AsynctronouslyInterruptedException. Si può quindi gestire l'interruzione nel catch dell'eccezione

La politica SkipStop

- InterrumpiblePeriodicThread esegue una busyWaitInterrompibile
- thresholdPolicyhandler modella un gestore che è in grado di cambiare strategia dopo un certo numero di deadline violate consecutivamente dallo stesso jo
- SkipStopPolicyHandler modella la politica in questione

La politica SkipStop

- Qui metti un diagramma di iterazione
- Quando si verifica un deadline miss bla bla

- Quando si esegue un job di recupero (va saltato come con la politica skip)

Risultati

- Metti a confronto due esecuzioni della stessa applicazione che mostra benefici nell’interrompere prima il job

Sviluppi futuri

- Estensione a sistemi multiprocessore
- Estensione a insiemi di processi sporadici

- Analisi di schedulabilità
- Politiche di gestione della parte non real-time dell'applicazione (pollingServer, deferrable server, Constant UtilizationServer ecc.)