Real Time Specification for Java

SUN Yunxia

LU Da

LI Haitao

Inhalt

- Einführung
- Eigenschaft von Java
- Lösungsansatz
- Literaturverzeichnis

Einführung

Java ist eine Sprache, und sich bereits in wichtigen Feldern gegen C++ durchgesetzt hat

- Die dem objektorientierten Ansatz folgt
- Es verfügt über einige Vorzüge
- Allerdings gibt es in den angespochenen Bereichen Anfordungen

Echtzeit-Verarbeitung

- Nicht unbedingt absoluter höherste Ausführungsgeschwindigkeit

- Unterscheidung:
  - harte Echtzeit (ab dem definierten Zeitpunkt sofort 0)
  - weiche Echtzeit (nicht schlagartig)

Eigenschaft von Java (1)

- Objektorientiert
- Einfach

  die Syntax leichter als C/C++
- Interpretiert
  1. Java-Byte-Code erzeugen
  2. durch JVM oder Java-Prozessoren interpretiert wird
  3. das Konzept des Interpreters erleichtert die Fehlersuchen

Eigenschaft von Java (2)

- Robust
  1. Speicherlöcher & Referenzen durch Garbage Collection vermieden werden
  2. Laufzeitfehler Ausnahmen erzeugen
- Plattformunabhängig & portable
  1. durch das Konzept des Java-Byte-Code interpretiert wird
  2. auf dem Zielsystem, ein angepasster Interpreter & einige Java-Basisklassen oder ein Java-Prozessor vorhanden sein

Eigenschaft von Java (3)

- Debenläufig

  die Theads anbietet

- Dynamisch

  1. durch das Nachladen von Programmteilen zur Laufzeit ermöglicht
  2. Dieses ist dadurch in der Lage, sich an veränderte Umgebungen anzupassen

Java´s Probleme mit Echtzeit

Probleme in 4 kritischen Bereichen

- Scheduling
- Speichermanagement
- Hardwarezugriffe
- Synchronisation

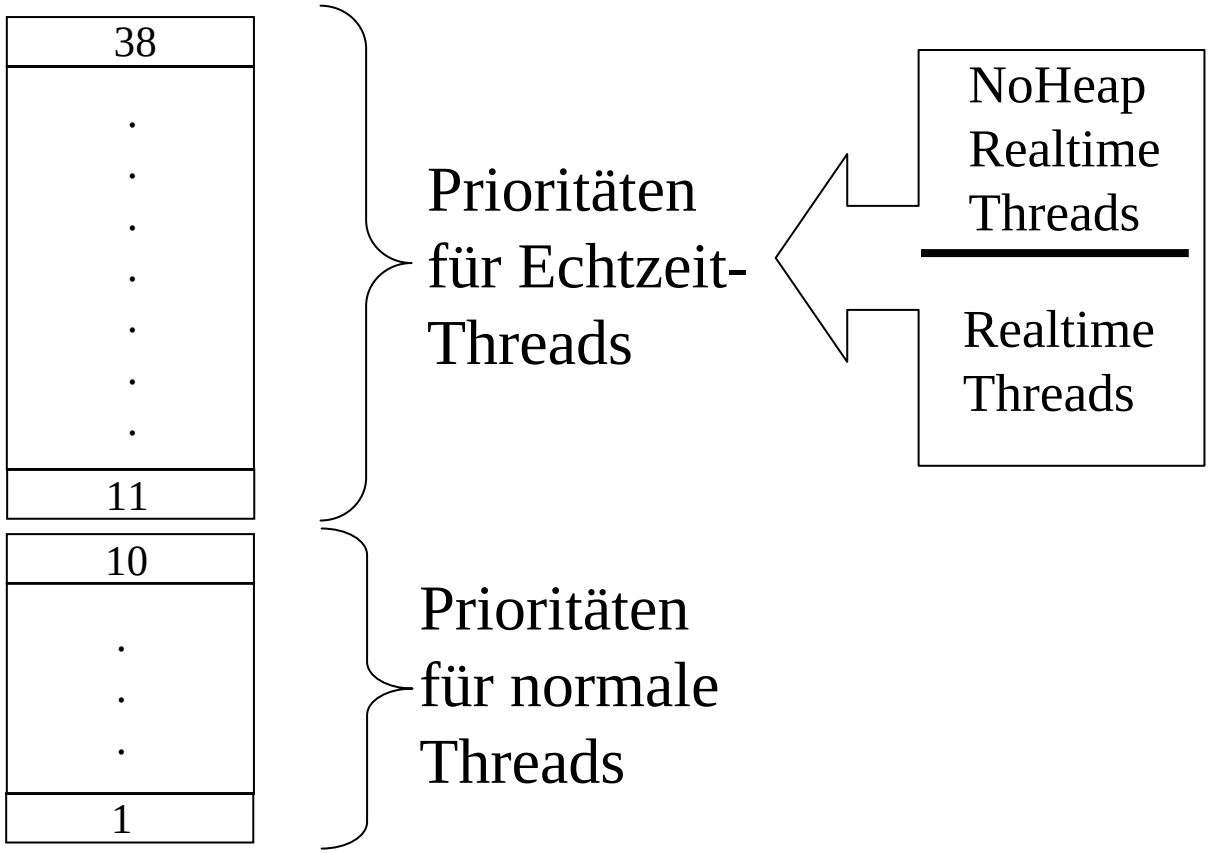
Lösungsansatz

- Scheduling
- Speicherverwaltung
- Synchronisation
- Asynchrones Event-Behandlung

Scheduling (1)

- Preemptiver Scheduler mit festen Prioritäten
- 28 Prioritätsstufen für Echtzeitthreads
- Zusätzlich 10 Prioritätsstufen für normale Java-Threads
- Mit höherer Priorität jene mit niedrigerer Priorität unterbrechen
- Zur Vermeidung von Prioritätsumkehr

Scheduling (2)



Scheduling (3)

Folgende Typen von Echtzeitthreads werden von RTSJ bereitgestellt:

- *RealtimeThread*

  Objekte auf dem Heap ansprechen und für die weniger zeitkritischen Aufgaben

- *NoHeapRealtimeThread*

  Keine Objekte auf dem Heap ansprechen und für die kleine zeitliche tolerante Aufgaben

- *AsyncEventHandler*

  Diese Art ist im Prinzip Äquivalent zu einem *Real-TimeThread* ausser dass sie an ein *AsyncEvent* gebunden ist.

Scheduling (4)

- Beispiel

```
import javax.realtime.*;
public class Hello1 {
   public static void main ( String[] args ) {
      RealtimeThread rt = new RealtimeThread() {
         public void run() {
            System.out.println("Hello RT world"); }};
      rt.start();
      try{
         rt.join(); }
      catch(InterruptedException e){}
      System.exit(0);
}}
```

Scheduling (5)

- Ändern der Thread-Priorität

```
...
int initialPriority;
PriorityParameters pp;
RealtimeThread me = RealtimeThread.currentRealtimeThread();

pp = (PriorityParameters)me.getSchedulingParameters();
initialPriority = pp.getPriority();
pp.setPriority(initialPriority + 1);
...
```

Speicherverwaltung (1)

- Keine GC(Garbage Collection)
- RTSJ kennt folgende Speichertypen:
  - *Heap memory*
  - *Immortal memory*
  - *Scoped memory*
- Zwei neue Speichertypen definiert

Speicherverwaltung (2)

- Immortal Memory
  - steht allen Threads gemeinsam zur Verfügung
  - Objekte hierin existieren bis zum Ende der Anwendung
  - kein GC
  - Objekte exisitieren auch weiter, wenn keine Referenzen auf sie mehr vorhanden sind

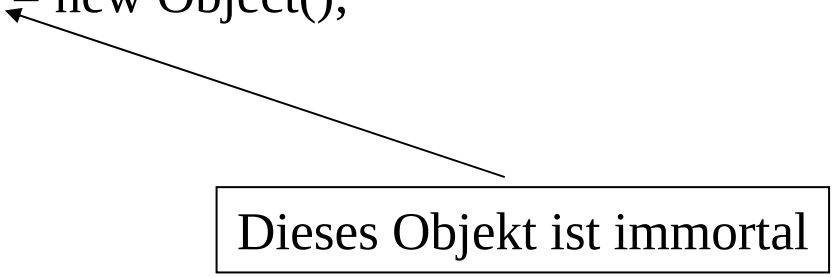
Speicherverwaltung (3)

- ScopedMemory
  - Speicherbereich mit begrenzter Lebensdauer
  - existiert, solange es Realtime-Threads mit einer Referenz hierauf gibt
  - muss vor Benutzung erzeugt werden
  - Unterscheidung :
    - LTMemory (Physical Linear Time Memory)
    - VTMemory (Physical Variable Time Memory)

Speicherverwaltung (4)

- Beispiel für Immortal-Objekt

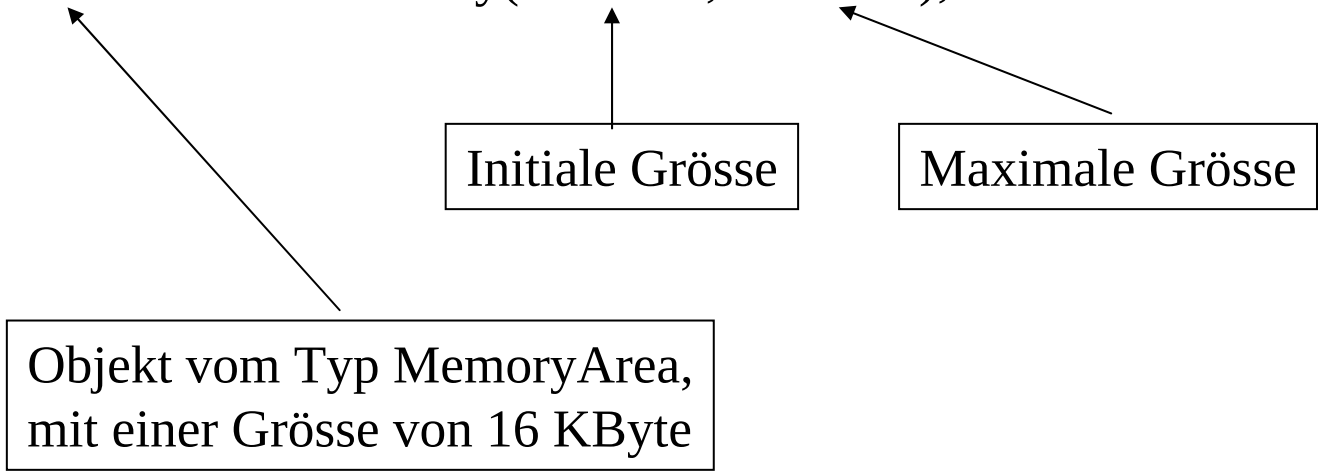
```
...
ImmortalMemory.instance().enter(
    new Runnable(){
        public void run() {
            o = new Object();
}});
...
```



Speicherverwaltung (5)

- Beispiel für ScopedMemory

```
LTMemory mem = new LTMemory(1024*16, 2048*16);
...
mem.enter(
...
);
```



Synchronisation (1)

- Zu Thema Synchronisation beschreibt die RTSJ unter anderem die drei Bereiche

1. Organisation von Threads
2. Prioritätsumkehr
3. nicht blockierende Kommnikation zwischen Threads

Synchronisation (2)

- Organisation von Threads

1. wenn Threads mit gleicher Priorität erlaubt sind
2. Threads werden in Warteschlangen für die jeweilige Priorität organisiert
3. Warteschlangen realiesieren FIFO-Prinzip

Synchronisation (3)

- Vermeidung von Prioritätsumkehr

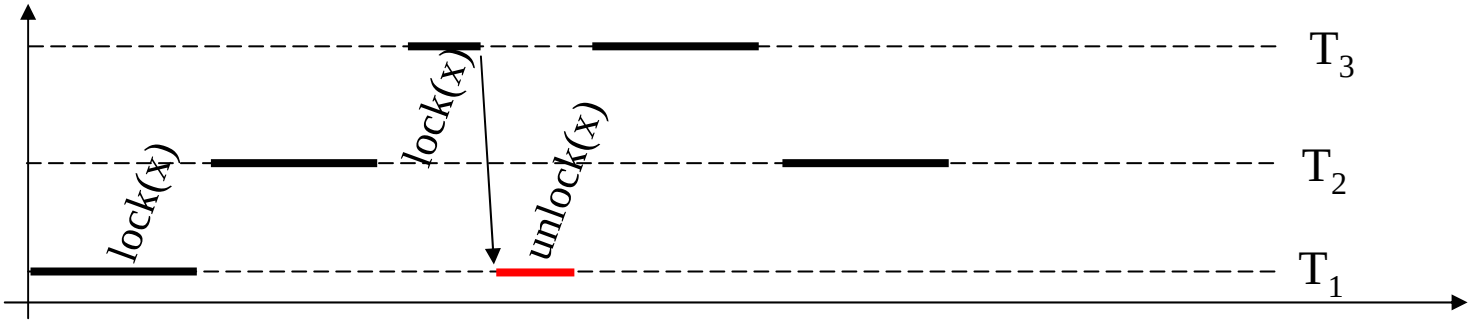
1. minimal : priority inheritance protocol

2. vorgesehen aber nicht gefordert : priority ceiling protocol

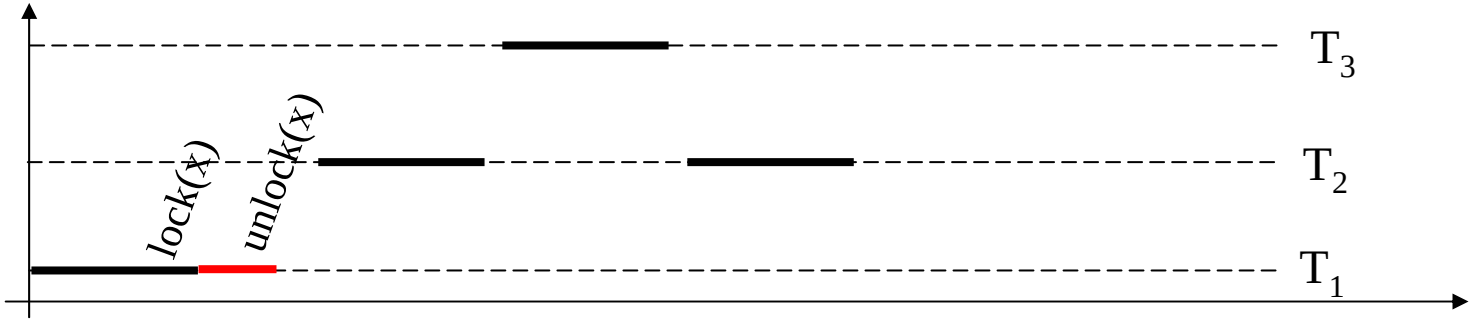
Synchronisation (4)

- Beispiele für priority inheritance und priority ceiling

priority inheritance



priority ceiling

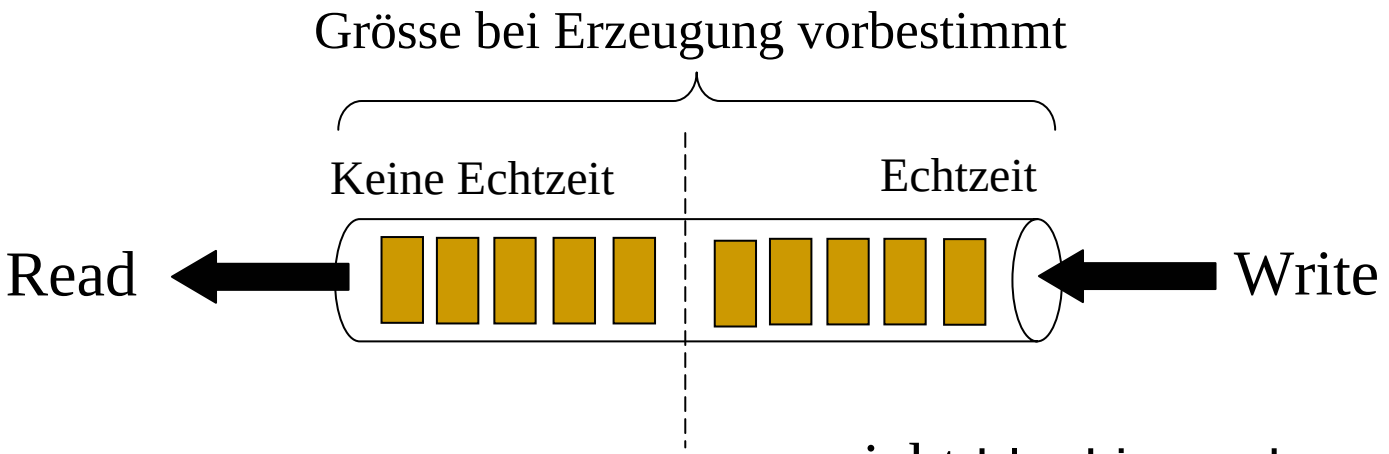


Synchronisation (5)

- Kommnikation zwischen NoheapRealtime-Threads und Realtime-Threads, normalen Java-Threads
  - nicht blockierende Kommnikationsmöglichkeit
    - wait-free queues erlauben Echtzeitthreads mit nomalen Thread zu Synchronisieren
  - Ziel : Beeinflussung des NoheapRealtime -Threads bei Synchronisation mit Realtime-Thread (normalen Threads) durch Garbage Collection verhindern

Synchronisation (6)

- Schema für wait free write queues



- blockierend
- synchronisiert

- nicht blockierend
- nicht synchronisiert
- wenn Queue voll ist, kann ein Element durch Write überschrieben werden

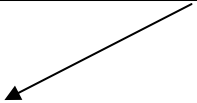
Asynchrone Event-Behandlung(1)

- Überwachung von internen (selbsProgrammierten) und externen (Hardware-Interrupts etc.) Ereignissen
- Bei Auftreten des Ereignisses : Auslösen von vorher definierten Aktionen
- Programmiertechnisch : 2 Klassen
  - ***AsyncEvent***
    symbolisiert das zu überwachende Ereignis.
  - ***AsyncEventHandler***
    enthält den auszuführenden Code

Asynchrone Event-Behandlung(2)

- Beispiel

Eigene Klasse (abgeleitet von AsyncEventHandler), die den Code enthält



AsyncEvent event = new AsyncEvent()

AsyncEventHandler handler = new SigHandler();

event.addHandler(handler); *//handler wird an event gebunden*

event.bindTo(“25”); *//event wird an Signal 25 gebunden*

Fazit(1)

- RTSJ

1. Änderungen an der JVM
2. Eventuell Änderungen am Compiler
3. Realisierung auf Realtime Betriebssystem

Fazit(2)

- Real Time Java wird nie die gleiche Perfomanz wie entsprechender C / C++ Code haben
- Real Time Java System wird nie so schlank wie entsprechender C / C++ - Code sein
- Real Time Java erbt Robustheit und Übersichtlichkeit von Java

Literaturverzeichnis

- Bollela Greg et al., The Real-Time Specification for Java, ADDISONWESLEY, Juni 2000
- Dibble, Peter C. : Real-Time JAVA Platform Programming , 2002 , Sun Microsystems Press
- Brich, Hinsken, Krause : Echtzeit-programmierung in Java , 2000 , Siemens
- Dr. Yerraballi Ramesh, Real-Time Java: A Specification, http://www.jconsortiom.org/rtjwg.html
- Esmertec Jbed, Homepage, http://www.esmertec.com

Literaturverzeichnis

- Bollela Greg et al., The Real-Time Specification for Java, ADDISONWESLEY, Juni 2000
- Dibble, Peter C. : Real-Time JAVA Platform Programming , 2002 , Sun Microsystems Press
- Brich, Hinsken, Krause : Echtzeit-programmierung in Java , 2000 , Siemens
- Dr. Yerraballi Ramesh, Real-Time Java: A Specification, http://www.jconsortiom.org/rtjwg.html
- Esmertec Jbed, Homepage, http://www.esmertec.com

Vielen Dank für Ihre Aufmerksamkeit!