Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Echtzeitprogrammierung in Java

Franziska Häger

Seminar: Prozesssteuerung und Robotik

Agenda

Einführung
- Warum Java?
- Java „Real Time“ Probleme
- Real Time Specification for JAVA (RTSJ)

Programmierung
- Threads
- Memory-Management
- Time, Timer & Clocks
- Asynchrone Ereignisse

Fazit

Warum Java?

Java:

- objektorientiert
- plattformunabhängig + portabel
- robust
- nebenläufig
- hoher Bekanntheitsgrad
- VM auf vielen Geräten schon vorhanden

Aber Java ist nicht als Echtzeit-Sprache ausgelegt und hat in diesem Bereich daher erheblich Mängel.

Java „Real Time“ Probleme

Thread Management:

Java Threads werden von der VM Initialisiert aber vom Betriebsystem gescheduled

-> keine Garantien für Prioritäten und Scheduling durch VM

Garbage Collection:

Die Garbage Collection in Java läuft unvorhersehbar und hält alle anderen Threads an.

-> unmöglich für Harte-Echtzeit

Java „Real Time“ Probleme

Class Loading

Java Specification verlangt, dass Klassen erst bei der ersten Verwendung initialisiert werden.

-> je nach Klasse und Speicherort verschiedene Wartezeiten

Compilation

Java Code wird interpretiert. Häufig ausgeführte Methoden werden jedoch von der VM zu nativem Code kompiliert.

-> interpretierter und nativer Code haben verschiedenes Zeitverhalten

-> Zeitpunkt des Kompilierens ist nicht vorhersagbar

Java „Real Time“ Probleme

Hardware nahe Programmierung

-> In Java sind keine direkten Hardwarezugriffe möglich.

Systemaktivitäten

Hoch priorisierte Systemaktivitäten können Java-Applikationen unterbrechen und Schwankungen im Zeitverhalten auslösen.

Real Time Specification for JAVA (RTSJ)

JSR1, 15. Dezember 1998: The Real Time Specification for Java

-> Januar 2002: RTSJ 1.0

- Erweiterung der Java Spezifikation um Echtzeitverarbeitung zu ermöglichen.

- Implementierungen:
    - IBM WebSphere Real Time
    - IBM/Apogee Aphelion
    - TimeSys RTSJ Reference Implementation
    - Sun Java Real-time (Java RTS)

Threads

Echtzeit- und „Normale-"Programmanteile sollen in einer Anwendung laufen können.

-> 2 Neue Threadtypen:

javax.realtime.RealtimeThread

Echtzeit-Thread, Priorität über Standard Thread

javax.realtime.NoHeapRealtimeThread

Echtzeit-Thread, höchst priorisiert (> GC),

kein Heap-Zugriff

Threads können periodisch, aperiodisch und sporadisch laufen-

-> Steuerung über entsprechende ReleaseParameter

Threads2

Scheduling mittels Scheduler

-> Standard Scheduler ist fixed-priority pre-emptive

-> eigene Scheduler Implementierungen möglich

-> priority inheritance, priority ceiling

JRTS verlangt mindestens 28 Prioritäten

-> Steuereung über PriorityParameters

DeadlineMissHandler, OverrunHandler

-> kann an ReleaseParameter übergeben werden

-> wird gerufen, wenn Deadline nicht eingehalten wurde bzw. Laufzeit überschritten wurde

Thread Beispiel1

```
Runnable runnable = new Runnable(){
        public void run() {
                System.out.println("Hello RT-World!");
        }
};
PriorityParameters prio =
new PriorityParameters(PriorityScheduler.instance().getMaxPriority());
PeriodicParameters period  =
new PeriodicParameters(new RelativeTime(1,0));
RealTimeThread rt t =
new RealtimeThread(sched,period,null,null,null,runnable);
```

Thread Beispiel2

```
int prio = PriorityScheduler.instance().getMaxPriority();
RelativeTime period = new RelativeTime(20,0);
NoHeapRealtimeThread nhrt = new NoHeapRealtimeThread(
                                    new PriorityParameters(prio),
                                    new PeriodicParameters( period),
                                            ImmortalMemory.instance()) {
        public void run() { . . . }
};
```

Scheduler Beispiel

```
RelativeTime rt = new RelativeTime(5000L, 0);
AperiodicParameters ap = new AperiodicParameters(rt, rt, null, null);
MemoryParameters mem= new MemoryParameters(60000L, 60000L);

PriorityScheduler ns = PriorityScheduler.instance();
Scheduler.setDefaultScheduler(ns);
PriorityParameters pp = new PriorityParameters(ns.getMinPriority()+10);
RealtimeThread rtt = new RealtimeThread(pp, ap);
boolean b = ns.setIfFeasible(rtt, ap, mem);
if(b)
        rtt.start();
```

Thread Communication

3 Wait-Free Queues zur Kommunikation zwischen normalen und Real Time Threads

- WaitFreeWriteQueue
  - blockierungsfreies schreiben, synchronisiertes Lesen
- WaitFreeReadQueue
  - blockierungsfreies lesen, synchronisiertes schreiben
- WaitFreeDequeue
  - blockierungsfreies lesen und schreiben und synchronisiertes lesen und schreiben

Memory-Management

3 Memory Areas

- Standard Heap
  - für alle Threads, GC
- Immortal Memmory
  - für alle Threads, kein GC
- Scoped Memory
  - nur für RT-Threads, kein GC, Freigabe wenn Task beendet
  - Subtypen: VTMemory, LTMemory

Memory-Management

Physical Memory

falls spezieller physikalischer Speicher benötigt wird (z.B. für DMA)

Zugriff auf physischen Speicher über RawMemoryAccess

Spezial NoHeapRealTimeThread

kein Zugriff auf Heap

Memory Beispiele1

```
ScopedMemory ma = new LTMemory(60000L);
Long size = ma.size();
ma.enter(new Runnable(){
        public void run(){
                …
        }
});

Object immortalObj =
ImmortalMemory.instance().newInstance(Object.class);
```

Memory Beispiele2

```
short s = 12;
long address = 015002;
long size = 2;

RawMemoryAccess rawMemory =
        new RawMemoryAccess(address, size);

rawMemory.setShort(0, s);
short s2 = rawMemory.getShort(0);
```

Time

HighResolutionTime

- Nanosekunden (Standard Java nur Millisekunden)

3 Subtypen

- AbsoluteTime
  - apsolute Zeit seit dem 01.01.1970 GMT
- RelativeTime
  - Zeit in Relation zu einem Startpunkt
- RationalTime
  - zur Darstellung vom Sequenzen und Perioden

Timer & Clock

2 Timer

- OneShotTimer
  - löst einmalig ein Event aus
- PeriodicTimer
  - löst in bestimmten intervallen ein Event aus

Timer benötigen eine Clock

- bestimmt Zeitbasis und länge eines Ticks
- per default System Real Time Clock

Timer Beipiel1

```
final Boolean stopLooping = false;
OneShotTimer timer2 = new OneShotTimer(
        new RelativeTime( 10000, 0),
        new AsyncEventHandler(){
                public void handleAsyncEvent(){
                        stopLooping = true;}
                });
timer.start();
while(!stopLooping){
        Thread.sleep(1000);
}
```

Timer Beipiel2

```
PeriodicTimer timer = new PeriodicTimer(
        null, //start now
        new RelativeTime( 1500, 0),  // tick = 1.5 s
        new AsyncEventHandler(){
                public void handleAsyncEvent(){
                        System.out.println("Tick");
                };
        });
timer.start();
Thread.sleep(20000);
```

Asynchrone Events

AsyncEvent

repärsentiert internes oder externes Ereignis

2 EventHandler

AsyncEventHandler

ähnlich wie RT-Thread, können aber nur geblockt oder gescheduled sein, nicht wartend oder schlafend

BoundAsyncEventHandler

an einen bestimmten Thread gebunden

Asynchronous Transfer of Control

AsynchronouslyInterruptedException

muss bei Unterbrechung von Methoden geworfen werden

wird durch throw oder interrupted() auf einem RealTimeThread ausgelöst

AsyncEventHandler kann Thread sauber beenden und handeln

Event Beispiele

```
AsyncEvent Interrupt = new AsyncEvent();
SchedulingParameters priParams = new
PriorityParameters(PriorityScheduler.instance().getMaxPriority());
ReleaseParameters releaseParams = new SporadicParameters(new
RelativeTime(1,0));
AsyncEventHandler InterruptHandler = new
        BoundAsyncEventHandler(
                priParams, releaseParams, null, null, null, false, null);
Interrupt.addHandler(InterruptHandler);
Interrupt.bindTo("0177760");

ReleaseParameters.setDeadlineMissHandler(handler);
```

Fazit

RTSJ Erweiterungen machen Java echtzeitfähig

Die Entwicklung von Software mit weichen und harten Echtzeitanforderungen ist möglich und wird bereits in der Praxis verwendet.

Implementierungen existieren bisher nur für RT-Linux und Solaris. Eine WindowsCE Implementierung ist geplant.

Beispiele

TemperatureControl Beispiel

http://www.devx.com/Java/Article/33475/1954?pf=true

Übungsaufgaben aus einem RT-Java Kurs

http://www.cs.wustl.edu/~mdeters/cait/2003/04/

Quellen

RealTime Specification for Java

http://www.rtsj.org/docs/rtsj_1.0.2_spec.pdf

An Introduction to Real-Time Java Technology

http://java.sun.com/developer/technicalArticles/Programming/rt_pt2/index.html

http://java.sun.com/developer/technicalArticles/Programming/rt_pt1/index.html

Go Inside the Java Real-Time System

http://www.devx.com/Java/Article/33475/1954?pf=true

Quellen

IBM Technical Papers: Real-time Java, Part 1 - 6

http://www.ibm.com/developerworks/java/library/j-rtj1/index.html?S_TACT=105AGX02&S_CMP=EDU

Real-Time Systems and Programming Languages

Von Alan Burns and Andy Wellings

Fragen?

Vielen Dank für Eure Aufmerksamkeit