Programmiersprachen für Echtzeitsysteme

Real-Time Java

Motivation

- Java ist eine sehr weit verbreitete Programmiersprache
- Vorteile:
  - Portabler Code durch virtuelle Maschine
  - Objektorientierte Paradigma
  - Strengere Typisierung
  - Einfacher Umgang mit Speicher (keine Zeiger, Garbage Collection)
- Nachteil: nicht echtzeitfähig (siehe nächste Folien)

→ RTSJ (Real-Time Specification for Java) **erweitert** Java:

  - Erweiterung der Spezifikation der Sprache Java
  - Erweiterung der Java Virtual Machine Spezifikation
  - Entwicklung einer Echtzeit-API

Design-Prinzipien RTSJ

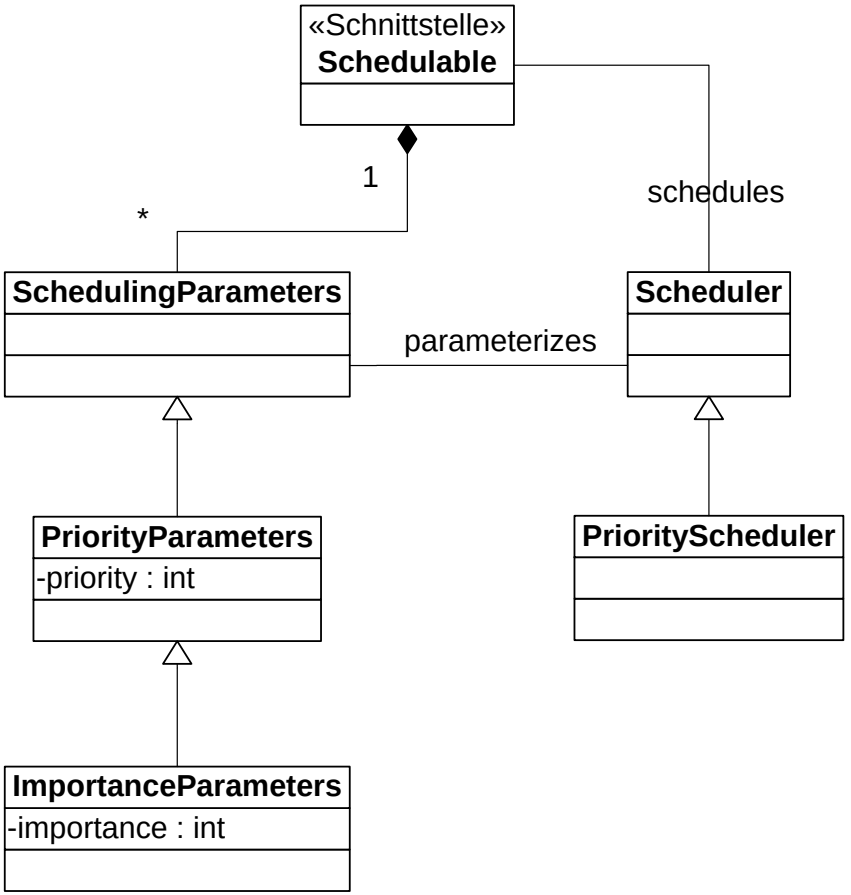
1. Keine Einschränkungen auf bestimmte Java-Umgebung (z.B. eine bestimmte Version von JDK)
2. Rückwärtskompatibilität
3. „Write once carefully, run anywhere conditionally“
4. Unterstützung aktueller Entwicklungsprozesse für Echtzeitsysteme
5. (Zeitlich) vorhersagbare Ausführung
6. Keine syntaktischen Erweiterungen
7. Kein Verbot von Implementierungsabweichungen (allerdings sollen diese sorgfältig dokumentiert werden).

Scheduling in Java

- Java Spec: „…threads with higher priority are generally executed in preference to threads with lower priority…"
- Scheduler:
  - Algorithmus nicht festgelegt
  - Keine vorgeschriebene Anzahl von Prioritäten
  - Verwendung von Round-Robin oder FIFO bei Prozessen gleicher Priorität nicht spezifiziert

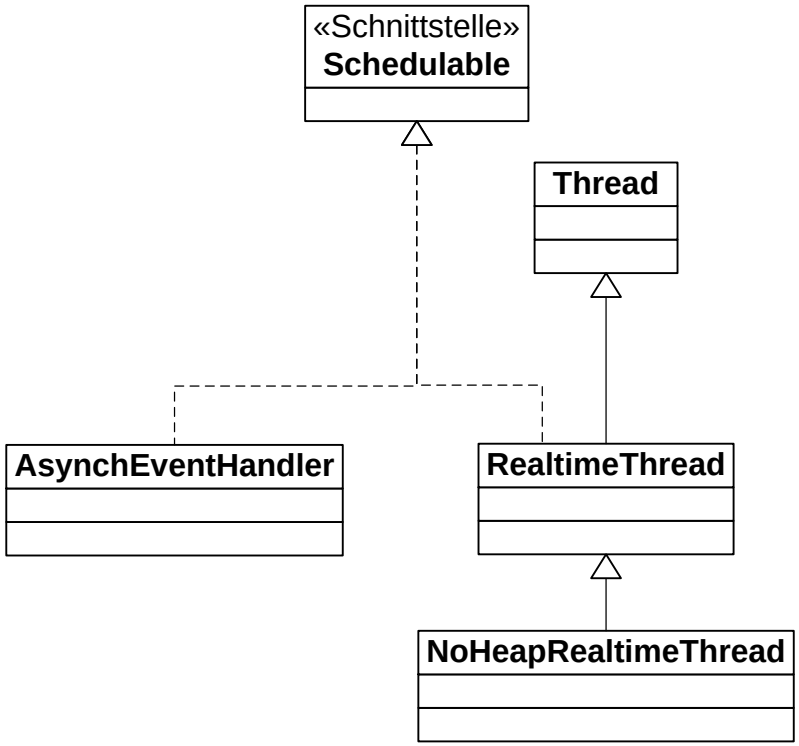
Scheduling in RTJS

- Einführung des PrioritySchedulers:
  - feste Prioritäten
  - präemptives Scheduling
  - mindestens 28 Prioritätsebenen für Echtzeitprozesse
- Unterschiedliche Parameter:
  - Schedulingparameter (traditionelle Priorität, importance-Feld für Überlastsituationen)
  - Freigabeparameter (Parameter für periodische, aperiodische, sporadische Prozesse)
  - Speicherparameter: definiert notwendigen Speicherplatz
  - Prozessgruppenparameter: zur Verwaltung einer Menge von aperiodischen oder sporadischen Prozesse als Meta-periodischen Prozess



Threads in RTJS

- `RealtimeThread`:
  - Kontrolliert durch den Scheduler.
  - Kann neben dem Heap auch eigenen Speicher benutzen.
  - Zugriff auf physikalischen Speicher möglich.
- `NoHeapRealtimeThread`:
  - Zugriff auf Objekte im Heap verboten.
  - Manipulation von Referenzen zu Objekten im Heap verboten.
  - Muss mit einem geschützten Speicherbereich erzeugt werden.
  - Kann den GarbageCollector unverzüglich unterbrechen.
- `AsynchEventHandler`:
  - realisiert Unterbrechungsbehandlungen



Speichermanagement in Java

- Der Garbage Collector ist einer der Hauptgründe, die gegen die Verwendung von Java in Echtzeitsystemen sprechen:
  - In regelmäßigen Abständen wird der Garbage Collector als Prozess im Hintergrund ausgeführt.
  - Der GC ermittelt diejenigen Objekte, auf die nicht mehr verwiesen wird. Diese Objekte werden markiert und in einem zweiten Durchgang entfernt.
  - Problem: Garbage Collector benötigt langwierige Ausführungszeiten und kann nicht unterbrochen werden.
- Ansatz in RTSJ: Veränderung des Begriffs der Lebenszeit
  - manuelle Steuerung: Kontrolle der Lebenszeit via Programmlogik.
  - automatische Steuerung: wie bisher über Sichtbarkeit der Objekte.

Speichermanagement in RTSJ (1)

- RTSJ unterscheidet zwischen vier Speicherarten:
  - Heap memory (Standardspeicher von Java):
    - Verwaltung erfolgt durch den Garbage Collector
  - Immortal memory
    - Wird durch alle RealtimeThreads gemeinsam benutzt.
    - Pro Instanz der Virtual Machine existiert genau ein solcher Bereich.
    - Der GarbageCollector hat auf den Bereich keinen Zugriff → alloziierte Objekte bleiben bis zum Ende der Ausführung der Virtual Machine im Speicher
    - Es existiert kein Mechanismus zur Freigabe von Objekten.

Speichermanagement in RTSJ (2)

- Fortsetzung:
  - Scoped memory (Speicher mit eingeschränktem Lebensraum)
    - Der Benutzer kann scoped memory Speicher manuell anlegen.
    - Dabei wird zwischen zwei Arten von Speicher unterschieden:
      - LTMemory: Objektallokationen sind in linearer Zeit durchführbar
      - VTMemory: es werden keine Zeitgarantien gegeben
    - Mit dem Schlüsselwort `enter` kann die Lebensdauer (Klammern begrenzen Lebensraum) definiert werden:
      ```
      myScopedMemArea.enter(){ … }
      ```
    - Alle mit `new` in dem Bereich erzeugten Objekte werden im `ScopedMemory` Bereich alloziiert.
  - RawMemoryAccess : Zusätzlich erlaubt RTSJ im Gegensatz zu Java auch Zugriff auf physikalischen Speicher durch Einführung der zusätzlichen Speicherart `RawMemoryAccess`

Weitere Ergänzungen

- Synchronisation:
  - Bei Monitoren muss jede RTSJ-Implementierung den Priority Inheritance Algorithmus implementieren.
  - Die Implementierung von Priority-Ceiling ist optional.
  - Zwischen Real-Time Threads und Standard Threads können Wait Free Queues (nicht blockierende Nachrichtenwarteschlangen) verwendet werden.
- Zeit:
  - Einführung der Klasse `Time` (mit den Unterklassen `AbsoluteTime`, `RelativeTime`)
  - Einführung der abstrakten Klasse `Clock`
  - Jede RTSJ-Implementierung muss die Klasse `RealtimeClock` enthalten.
  - Einführung der Klasse `Timer` (Unterklassen `OneShotTimer`, `PeriodicTimer`)

Programmiersprachen für Echtzeitsysteme

Zusammenfassung

Was Sie aus diesem Kapitel mitgenommen haben sollten

- Kriterien bei der Auswahl der Sprache sind:
  - Sicherheit
  - Komfort bei der Entwicklung (v.a. Existenz geeigneter Entwicklungswerkzeuge)
  - projektierbares Zeitverhalten
  - Möglichkeit zur hardwarenahen, nebenläufigen Programmierung
  - Portabilität
- Zur Sicherheit tragen eine strenge Typisierung und Prüfungen zur Laufzeit bei.
- Zur Erhöhung der Portabilität werden hardwareabhängige und –unabhängige Codeteile häufig getrennt.
- Mechanismen zum Prozessmanagement / -synchronisation und dem Umgang mit Zeit erleichtern die Programmierung von Echtzeitsystemen
- Virtuelle Laufzeitumgebungen (wie z.B. Virtual Machine in RTSJ) eignen sich nur bedingt zur Verwendung in Echtzeitsystemen
- Kenntnis über dieKonzepte der besprochenen Programmiersprachen, die über Standardsprachen wie C bzw. JAVA hinausgehen
- Trotz der Existenz vieler komfortabler und sicherer Programmiersprachen für Echtzeitsysteme wird der überwiegende Teil der Echtzeitsysteme in C (plus POSIX) entwickelt.

Klausur WS 07/08 (8 Punkte)

- Erläutern Sie die Konzepte von Ada zur Unterstützung von Unterbrechungen (Interrupts) und Prozesssynchronisation.
- Erläutern Sie die Konzepte zur Speicherverwaltung in Real-Time Java.
- Antwort: siehe Folien

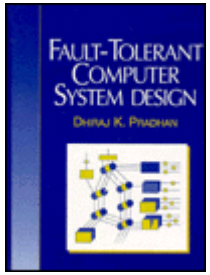
Kapitel 9

Entwicklung von sicherheitskritischen Systemen

Inhalt

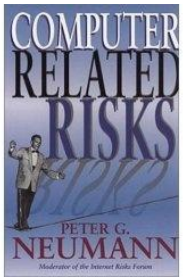
- Einleitung / Motivation / Definitionen
- Zertifizierungsstandards
- Sicherheitsanalyse & ASIL-Einstufung
- Analyse der möglichen Fehler
- Fehlererkennung
- Fehlertoleranzmechanismen
- Anforderungen an die Softwareentwicklung

Literatur

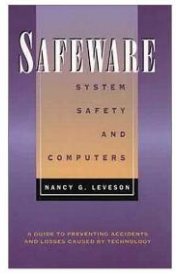


Dhiraj K. Pradhan: Fault-Tolerant Computer System Design, Prentice Hall 1996

W.A.Halang, R.Konakovsky: Sicherheitsgerichtete Echtzeit-systeme, Oldenburg 1999



Peter G.Neumann: Computer Related Risks, ACM Press 1995



Nancy G.Leveson: Safeware, Addison-Wesley 1995

Klaus Echtle: Fehlertoleranzverfahren, Springer-Verlag 1990 (elektronisch unter http://dc.informatik.uni-essen.de/Echtle/all/buch ftv/)

http://www.system-safety.org/

Entwicklung sicherheitskritischer Systeme

Negativbeispiele (Motivation)

Sicherheit fängt schon im Kleinen an

- Lexikalische Konventionen können Fehler verhindern.
- Negatives Beispiel: FORTRAN
  - In FORTRAN werden Leerzeichen bei Namen ignoriert.
  - Variablen müssen in FORTRAN nicht explizit definiert werden
- Problem in Mariner 1:
  Aus einer Schleife

```
DO 5 K = 1,3
```

  wird durch versehentliche Verwendung eines Punktes

```
DO5K=1.3
```

  eine Zuweisung an eine nicht deklarierte Variable.
  → Zerstörung der Rakete, Schaden 18,5 Millionen $

Ariane 5 (1996)

- Selbstzerstörung bei Jungfernflug:
- Design:
  - 2 redundante Meßsysteme (identische Hardware und Software) bestimmen die Lage der Rakete (hot-standby)
  - 3-fach redundante On-Board Computer (OBC) überwachen Meßsysteme
- Ablauf:
  - Beide Meßsysteme schalten aufgrund eines identischen Fehlers ab
  - OBC leitet Selbstzerstörung ein
- Ursache:
  - Wiederverwendung von nicht-kompatiblen Komponenten der Ariane 4 (Speicherüberlauf, weil Ariane 5 stärker beschleunigt)

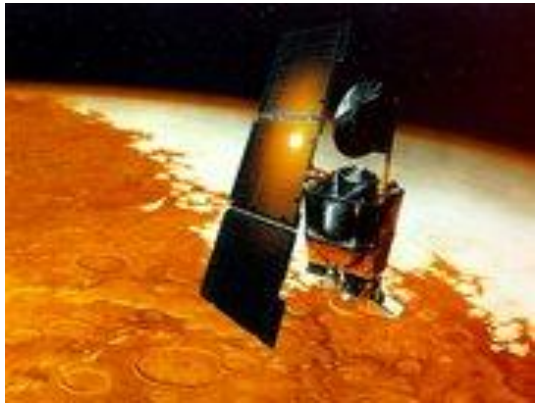
Weitere Informationen unter
http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html

Therac-25 (1985-1987)

- Computergesteuerter Elektronenbeschleuniger zur Strahlentherapie
- Das System beinhaltete 3 schwere Mängel:
  - Sicherheitsprüfungen im Programm wurden durch einen Softwarefehler bei jeder 64. Benutzung ausgelassen (wenn ein 6-bit Zähler Null wurde).
  - Behandlungsanweisungen konnten mittels Editieren am Bildschirm so abgeändert werden, dass die Maschine für die nächste Behandlung nicht den gewünschten Zustand einnahm (nämlich Niederintensität).
  - Mehrere Sicherheitsverriegelungen, die beim Vorgängermodell Therac-20 in Hardware realisiert waren, wurden nicht übernommen, sondern durch Software ersetzt.
- Folgen:
  - Mehrere Patienten erhielten anstatt der vorgesehenen Dosis von 80-200 rad Strahlungsdosen von bis zu 25000 rad (mehrere Tote und Schwerverletzte).
- Weitere Informationen unter http://sunnyday.mit.edu/papers/therac.pdf

Mars Climate Orbiter (1998)

- Verglühen beim Eintritt in die Atmosphäre
- Ursache:
  - Verwendung von unterschiedlichen Maßeinheiten (Zoll, cm) bei der Implementierung der einzelnen Komponenten.
  - Mangelnde Erfahrung, Überlastung und schlechte Zusammenarbeit der Bodenmannschaften



Weitere Informationen unter http://mars.jpl.nasa.gov/msp98/orbiter/

Explosion einer Chemiefabrik (1992)

- Explosion einer holländischen Chemiefabrik aufgrund eines Bedienfehlers
- Ablauf:
  - Computergesteuertes Mischen von Chemikalien.
  - Operateur (in Ausbildung) verwechselt beim Eintippen eines Rezeptes 632 (Harz) mit 634 (Dicyclopentadien).
- Folgen:
  - Explosion fordert 3 Menschenleben, Explosionsteile finden sich noch im Umkreis von 1 km.