

KV Betriebssysteme

Threads (I)

KV Betriebssysteme

Einheit 8: Threads (I)

Roland A. Eggersberger

Inst. f. Informationsverarbeitung und Mikroprozessortechnik (FIM)
Johannes Kepler Univ. Linz, Altenbergerstr. 69, A-4040 Linz, Austria

Inhalt

Main Thread
Threads erzeugen
Eigenschaften
Scheduling
Synchronisation
Kommunikation
Beispiele

KV Betriebssysteme

Threads (I)

Main Thread

- Thread der `main` ausführt
- Immer vorhanden

```
import java.io.*;
public class ThreadOne {
    :
    static public void main(String[] args) {
        :
    }
}
```

Threads erzeugen (I)

- Subklasse von `Thread` deklarieren
- Überschreiben von `run` mit den jeweiligen Aufgaben
- Aufruf von `start`, dadurch wird `run` ausgeführt

KV Betriebssysteme

Threads (I)

Threads erzeugen (II)

```
import java.io.*;
public class ThreadTwo extends Thread {
    public void run() {
        :
    }
    static public void main(String[] args) {
        Thread t = new ThreadTwo();
        t.start();
    }
}
```

Threads erzeugen (III)

- Methoden

`currentThread` - gibt den aktuellen Thread zurück

`sleep` - Unterbrechung der Ausführung (Zeit in ms)

`isAlive` - ob aktueller Thread lebt

KV Betriebssysteme

Threads (I)

Eigenschaften (I)

- Name
Abfragen mit `getName`
Auch setzen möglich (`setName`)
Üblich ist - `main` bzw. `Thread-n`

Eigenschaften (II)

- Priorität
Wird vom erzeugenden Thread geerbt
Veränderbar (`setPriority`)
Abfragen mit `getPriority`
Im Bereich von `Thread.MIN_PRIORITY` bis
`Thread.MAX_PRIORITY`

KV Betriebssysteme

Threads (I)

Eigenschaften (III)

- Threadgruppen
 - Gemeinsame Eigenschaften
 - Repräsentiert via **ThreadGroup**
 - getThreadGroup** liefert diese
- Dämons
 - Threads im Hintergrund
 - Setzen mittels **setDaemon** vor dem Starten
 - Abfragen via **isDaemon**

Eigenschaften (IV)

- Zustand
 - Erzeugt - nach **new**
 - Aktivierbar (I) - nach **start**
 - Nicht aktivierbar - I/O, **sleep**, **join**, **wait**
 - Aktivierbar (II) - I/O zu Ende, **sleep** zu Ende, Terminierung (auf **join**), **notify** (auf **wait**)
 - Aktivierbar (III) - auf **yield** (vom aktiven Zustand)
 - Terminiert - **run** beendet, Exception (nicht behandelt)

KV Betriebssysteme

Threads (I)

Scheduling (I)

- Thread-Scheduler - Zuteilung für aktivierbare Threads
- Prioritäten - Werden berücksichtigt, jedoch nicht absolut
- Time-Slicing - Unterbrechung aktiver Threads, wenn ihre Zeitscheibe abgelaufen ist

Scheduling (II)

- Problem - Kompakte Schleifen
 - `yield` in Schleifen, die sonst nicht (vom aktiven) in den aktivierbaren Zustand wechseln würden
 - `sleep` bewirkt sogar ein Wechseln in den nicht aktivierbaren Zustand

KV Betriebssysteme

Threads (I)

Synchronisation (I)

- Regelung des Zugriffs auf Objekte (bzw. deren Instanzvariablen)
- Synchronized Methoden
 - Zu jedem Objekt existiert eine Sperre (Monitor)
 - Müssen Threads erwerben, um synchronized Methoden auszuführen
 - Sperre wird erst beim Beenden der Methode freigegeben

Synchronisation (II)

- Auch für **static** Methoden - Monitor des Klassenobjektes
- Synchronized Anweisung
 - Synchronized wird auf einen Ausdruck angewendet, der ein Objekt angibt

KV Betriebssysteme

Threads (I)

Kommunikation (I)

- Warten auf Terminierung
Methode `join`
- Beliebiges Warten (siehe Thread-Zustände)
In synchronized Methoden
Bei `wait` wird die Sperre freigegeben, der Thread nicht aktivierbar
Bei `notify` (`notifyAll`) wird ein (alle) wartender Thread aktivierbar

Kommunikation (II)

- Typische Situation
Thread A macht `wait`, um auf ein Ergebnis von Thread B zu warten, dieser macht ein `notify` sobald er fertig ist
- Deadlock-Gefahr - Es passiert sehr leicht, dass ein Thread ein `notify` macht, obwohl der andere Thread das entsprechend geplante `wait` noch nicht erreicht hat. Hier helfen oft auch genau ausgetüftelte Prioritäten nichts.

KV Betriebssysteme

Threads (I)

Kommunikation (III)

- Tatsächlicher Deadlock

Zwei Threads warten auf das **notify** des jeweils anderen

Ein Thread wartet auf das **notify** eines terminierten Threads

Kommunikation (IV)

- Eigenschaften

Zusätzlicher Aufwand - Programm wird langsamer

Welcher wartende Thread bei einem **notify** fortgesetzt wird, ist nicht definiert

KV Betriebssysteme

Threads (I)

Beispiele

- Dämon-Threads
- Deadlocks
- Synchronized
- Race Conditions
- Time Slicing

Dämon-Threads

```
import java.io.PrintWriter;
public class Counter extends Thread {
    static PrintWriter out = new PrintWriter(System.out, true);
    public void run() {
        for(int i = 0; i < 4; i++) {
            out.println(getName() + ": " + i);
            try {
                sleep(1000);
            }
            catch(InterruptedException e) {
                out.println(e);
            }
        }
    }
}
```

KV Betriebssysteme

Threads (I)

```
static public void main(String args[]) {  
    out.println("Programmstart\n\n");  
    Thread t1 = new Counter();  
    Thread t2 = new Counter();  
    t1.setDaemon(true);  
    t2.setDaemon(true);  
    t1.start();  
    t2.start();  
    try{  
        sleep(2000);  
        t1.join();  
        t2.join();  
    } catch(InterruptedException e){  
        out.println(e);  
    }  
    out.println("\n\nProgrammende");  
}
```

Ausgabe

Programmstart
Thread-2: 0
Thread-3: 0
Thread-2: 1
Thread-3: 1
Programmende
Application terminated

Ausgabe ohne setDaemon

Programmstart
Thread-2: 0
Thread-3: 0
Thread-2: 1
Thread-3: 1
Programmende
Thread-2: 2
Thread-3: 2
Thread-2: 3
Thread-3: 3
Application terminated

KV Betriebssysteme

Threads (I)

```
Ausgabe erstes setDaemon  
auskommentiert  
Programmstart  
Thread-2: 0  
Thread-3: 0  
Thread-2: 1  
Thread-3: 1  
Programmende  
Thread-2: 2  
Thread-3: 2  
Thread-2: 3  
Thread-3: 3  
Application terminated
```

```
Ausgabe mit join  
Programmstart  
Thread-2: 0  
Thread-3: 0  
Thread-2: 1  
Thread-3: 1  
Thread-2: 2  
Thread-3: 2  
Thread-2: 3  
Thread-3: 3  
Programmende  
Application terminated
```

Leere Folie

KV Betriebssysteme

Threads (I)

Deadlocks

```
import java.io.PrintWriter;
public class Deadlock extends Thread {
    static PrintWriter out = new PrintWriter(System.out, true);
    static int a = 0;
    static int b = 0;
    public Deadlock(String name) {super(name);}
    public void run() {
        if(a == 0) {
            a = 1;
            for(long i = 0; i < 10000; i++);
            while(b != 0);
            // Hierher kommt man nie!
            a = 0;
        }
    }
}
```

```
if(b == 0) {
    b = 1;
    for(long i = 0; i < 10000; i++);
    while(a != 0);
    // Hierher kommt man nie!
    b = 0;
}
}

static public void main(String args[]) {
    out.println("Programmstart\n\n");
    Deadlock t1 = new Deadlock("Thread 1");
    Deadlock t2 = new Deadlock("Thread 2");
}
```

KV Betriebssysteme

Threads (I)

```
t1.start();
t2.start();
try {
    t1.join();
    t2.join();
}
catch(InterruptedException e) {
    out.println(e);
}
out.println("\n\nProgrammende");
}
```

Leere Folie

KV Betriebssysteme

Threads (I)

Leere Folie

Leere Folie

KV Betriebssysteme

Threads (I)

Synchronized

```
public class Konto {  
    private int Kontostand = 0;  
  
    public Konto (int stand) {  
        Kontostand = stand;  
    }  
  
    public synchronized void ueberweise(int betrag, Konto ziel){  
        synchronized(ziel) {  
            this.abheben(betrag);  
            Thread.yield(); // Umschalten erzwingen!  
            ziel.einzahlen(betrag);  
        }  
    }  
}
```

```
public synchronized void abheben(int betrag) {  
    // Unbegrenzter Überziehungsrahmen  
    Kontostand -= betrag;  
}  
  
public synchronized void einzahlen(int betrag) {  
    Kontostand += betrag;  
}  
  
public int Saldo() {  
    return Kontostand;  
}  
}
```

KV Betriebssysteme

Threads (I)

```
import java.io.PrintWriter;

public class Ueberweisungstest extends Thread {
    static PrintWriter out = new PrintWriter(System.out, true);
    public static Konto a = new Konto(100);
    public static Konto b = new Konto(100);

    public void run() {
        out.println("Vor Ueberw.: a hat " + a.Saldo() + " Euro");
        out.println("Vor Ueberw.: b hat " + b.Saldo() + " Euro");
        a.ueberweise(50, b);
        out.println("Nach Ueberw.: a hat " + a.Saldo() + " Euro");
        out.println("Nach Ueberw.: b hat " + b.Saldo() + " Euro");
    }
}
```

```
static public void main(String args[]) {
    out.println("Programmstart\n\n");
    Thread t = new Ueberweisungstest();
    t.start();
    Thread.yield();
    out.println("Abfrage: a hat " + a.Saldo() + " Euro");
    out.println("Abfrage: b hat " + b.Saldo() + " Euro");
    try{
        t.join();
    }
    catch(InterruptedException e) {
        out.println(e);
    }
    out.println("\n\nProgrammende");
}
```

KV Betriebssysteme

Threads (I)

Ausgabe

Programmstart

Vor Ueberw.: a hat 100 Euro

Vor Ueberw.: b hat 100 Euro

Abfrage: a hat 50 Euro

Abfrage: b hat 100 Euro

Nach Ueberw.: a hat 50 Euro

Nach Ueberw.: b hat 150 Euro

Programmende

Application terminated

Leere Folie

KV Betriebssysteme

Threads (I)

Race Conditions

```
import java.io.PrintWriter;
public class Race extends Thread {
    static PrintWriter out = new PrintWriter(System.out, true);
    static Integer number = new Integer(0);

    public Race (String name) {
        super(name);
    }

    public void run() {
        int n = number.intValue();
        for(long i = 0; i < 1000000; i++); // Do something
        number = new Integer(n + 1);
    }
}
```

```
static public void main(String args[]) {
    out.println("Programmstart\n\n");
    Race t1 = new Race("Thread 1");
    Race t2 = new Race("Thread 2");
    t1.start();
    t2.start();
    try{
        t1.join();
        t2.join();
    }
    catch(InterruptedException e) {
        out.println(e);
    }
    out.println(number); // Ergebnis ausgeben
    out.println("\n\nProgrammende");
}
```

KV Betriebssysteme

Threads (I)

Ausgabe

```
Programmstart  
1  
Programmende
```

Geänderte Version

```
public void run() {  
    synchronized(number) {  
        int n = number.intValue();  
        for(long i = 0; i < 1000000; i++); // Do something  
        number = new Integer(n + 1);  
    }  
}
```

Ausgabe der geänderten Version

```
Programmstart  
2  
Programmende
```

Time Slicing

```
import java.io.PrintWriter;  
public class TimeSlice extends Thread {  
    static PrintWriter out = new PrintWriter(System.out, true);  
    public TimeSlice(String name) { super(name); }  
    public void run() {  
        out.println(getName() + " gestartet.");  
        for(int i = 1; i < 5; i++) {  
            long t = System.currentTimeMillis() + 1000;  
            while(System.currentTimeMillis() < t) ;  
            // Achtung: Hier kein sleep sondern dauernde Berechnung!  
            out.println(getName() + ": nach " + i + " Sekunden");  
        }  
        out.println(getName() + " beendet.");  
    }  
}
```

KV Betriebssysteme

Threads (I)

```
static public void main(String args[]){  
    System.out.println("Programmstart\n\n");  
    Thread[] t = new Thread[4];  
  
    for(int i = 0; i < 4; i++){  
        t[i] = new TimeSlice("TimeSlice-Thread " + i);  
        // t[i].setPriority(Thread.MIN_PRIORITY + i);  
        // t[i].setPriority(Thread.MIN_PRIORITY + 4 - i);  
        t[i].start();  
        try{  
            sleep(100);  
        }  
        catch(InterruptedException e){  
            out.println(e);  
        }  
    }  
}
```

```
try{  
    sleep(5000);  
}  
catch(InterruptedException e){  
    out.println(e);  
}  
}  
}
```

KV Betriebssysteme

Threads (I)

```
Gleiche Prioritäten
Programmstart
TimeSlice-Thread 0 gestartet.
TimeSlice-Thread 1 gestartet.
TimeSlice-Thread 2 gestartet.
TimeSlice-Thread 3 gestartet.
TimeSlice-Thread 0: nach 1 Sekunden
TimeSlice-Thread 1: nach 1 Sekunden
TimeSlice-Thread 2: nach 1 Sekunden
TimeSlice-Thread 3: nach 1 Sekunden
TimeSlice-Thread 0: nach 2 Sekunden
TimeSlice-Thread 1: nach 2 Sekunden
TimeSlice-Thread 2: nach 2 Sekunden
TimeSlice-Thread 3: nach 2 Sekunden
TimeSlice-Thread 0: nach 3 Sekunden
TimeSlice-Thread 1: nach 3 Sekunden
TimeSlice-Thread 2: nach 3 Sekunden
TimeSlice-Thread 3: nach 3 Sekunden
```

```
TimeSlice-Thread 0: nach 4 Sekunden
TimeSlice-Thread 0 beendet.
TimeSlice-Thread 1: nach 4 Sekunden
TimeSlice-Thread 1 beendet.
TimeSlice-Thread 2: nach 4 Sekunden
TimeSlice-Thread 2 beendet.
TimeSlice-Thread 3: nach 4 Sekunden
TimeSlice-Thread 3 beendet.
Application terminated
```

KV Betriebssysteme

Threads (I)

```
Steigende Prioritäten
Programmstart
TimeSlice-Thread 0 gestartet.
TimeSlice-Thread 1 gestartet.
TimeSlice-Thread 2 gestartet.
TimeSlice-Thread 3 gestartet.
TimeSlice-Thread 2: nach 1 Sekunden
TimeSlice-Thread 3: nach 1 Sekunden
TimeSlice-Thread 2: nach 2 Sekunden
TimeSlice-Thread 3: nach 2 Sekunden
TimeSlice-Thread 0: nach 1 Sekunden
TimeSlice-Thread 1: nach 1 Sekunden
TimeSlice-Thread 3: nach 3 Sekunden
TimeSlice-Thread 2: nach 3 Sekunden
TimeSlice-Thread 3: nach 4 Sekunden
TimeSlice-Thread 3 beendet.
TimeSlice-Thread 2: nach 4 Sekunden
TimeSlice-Thread 2 beendet.
```

```
TimeSlice-Thread 0: nach 2 Sekunden
TimeSlice-Thread 1: nach 2 Sekunden
TimeSlice-Thread 0: nach 3 Sekunden
TimeSlice-Thread 1: nach 3 Sekunden
TimeSlice-Thread 0: nach 4 Sekunden
TimeSlice-Thread 0 beendet.
TimeSlice-Thread 1: nach 4 Sekunden
TimeSlice-Thread 1 beendet.
Application terminated
```

KV Betriebssysteme

Threads (I)

```
Fallende Prioritäten
Programmstart
TimeSlice-Thread 0 gestartet.
TimeSlice-Thread 0: nach 1 Sekunden
TimeSlice-Thread 0: nach 2 Sekunden
TimeSlice-Thread 0: nach 3 Sekunden
TimeSlice-Thread 1 gestartet.
TimeSlice-Thread 2 gestartet.
TimeSlice-Thread 0: nach 4 Sekunden
TimeSlice-Thread 0 beendet.
TimeSlice-Thread 3 gestartet.
TimeSlice-Thread 2: nach 1 Sekunden
TimeSlice-Thread 1: nach 1 Sekunden
TimeSlice-Thread 1: nach 2 Sekunden
TimeSlice-Thread 2: nach 2 Sekunden
TimeSlice-Thread 1: nach 3 Sekunden
TimeSlice-Thread 2: nach 3 Sekunden
TimeSlice-Thread 1: nach 4 Sekunden
```

```
TimeSlice-Thread 1 beendet.
TimeSlice-Thread 2: nach 4 Sekunden
TimeSlice-Thread 2 beendet.
TimeSlice-Thread 3: nach 1 Sekunden
TimeSlice-Thread 3: nach 2 Sekunden
TimeSlice-Thread 3: nach 3 Sekunden
TimeSlice-Thread 3: nach 4 Sekunden
TimeSlice-Thread 3 beendet.
Application terminated
```