Threads

Threads

s. auch Material zu Threads in Informatik II

Warum parallel evtl. besser als sequentiell?

- mehrere SpielServer oder SpielClients
- Scheduler im Hintergrund
- Echtzeitanwendung
- nicht-verlässliches Netzwerk

Beispiel

Notifikation mehrerer Clients

```
for (Client c: clients) {
  try {
      c.macheZug(z);
  } catch (RemoteException e) {
      ...
  }
}
```

Warum schlecht?

Beispiel

Wie wartet man 3,5 Sekunden (z.B. im Playback)?

```
Calendar now = Calendar.getInstance();
int ms = now.get(Calender.MILLISECONDS);
while (now.get(Calendar.MILLISECONDS) - ms < 3500) {}
```

Warum schlecht?

Die Klasse `Thread`

Objekte vom Typ `Thread` sind Prozesse, die parallel ablaufen können

wichtige Methoden:

- `run()`, um den Prozess laufen zu lassen
  sollte überschrieben, aber nicht aufgerufen werden
- `start()`, um den Prozess zu starten
  ruft standardmässig nach Erzeugen des Prozesses `run()` auf
- `sleep(int ms, int ns)`, neu in Java 1.5
  hält Prozess für `ms` Millisekunden + `ns` Nanosekunden an

Alternative: Interface `Runnable` (nur Methode `run()`)

Problem mit parallelen Prozessen

nicht-atomare Kommandos,

Bsp. Server mit Anmeldung per eindeutigem Namen

```
public boolean anmelden(String n) {
  for (String s: namen) {
    if (s.equals(n)) {
       return false;
    }
  }
  namen.add(n);
  return true;
}
```

Was ist das Problem?

Bisherige Lösung

```
public synchronized boolean anmelden(String n)
```

jedes Objekt hat einen *Lock* (boolean)

das Ausführen von synchronized Methoden setzt den Lock

synchronized Methoden können nur ausgeführt werden, wenn der Lock frei ist

Nachteil: Objekt wird gesperrt, nicht einzelne Methoden!

Semaphore

Semaphor s ist Paar *(int c,Queue w)* mit 2 Operationen

- $P(s)$ =

```
if (c>0) {
   c=c-1;
} else {
   w.add(this);
   wait;
}
```

- $V(s)$ =

```
if (w.isEmpty()){
   c=c+1;
} else {
   w.get(0).continue();
   w.remove(0);
}
```

Semaphore benutzen

Vorteil von Semaphoren: flexibler gegenüber `synchronized`

kein Lock auf Objekten sondern auf Codebereichen

$P(s)$;
... *[kritischer Bereich]* ... ;
$V(s)$;

Anfangswert von c: Anzahl der Prozesse, die gleichzeitig kritischen Bereich betreten dürfen

Die Klasse `java.util.concurrent.Semaphore`

Konstruktor:

- `Semaphore(int c)`

Methoden:

- $P(s)$:
  `void acquire() throws InterruptedException`

- $V(s)$:
  `void release() throws InterruptedException`

Das Package `java.util.concurrent`

neu in Java 1.5

noch weitere Tools, Datenstrukturen, etc. für parallele Programme, z.B.

- `ConcurrentHashMap<K,V>`
- `ConcurrentLinkedQueue<E>`
- ...

Beenden von Threads

Methoden `stop`, `destroy` deprecated

Thread beenden durch

- Methode `run()` abarbeiten
- Exception über `run()` hinaus werfen

Beenden eines Threads z.B. signalisieren durch

- `void interrupt()`
- `boolean isInterrupted()`