Real-time faciliteiten

Wat willen we met tijd?

- Gebruik van de **tijd**.
  - tijd(sduur) meten.
  - (tot een) bepaalde tijd slapen.
  - beperkte tijd wachten = time-outs.
- **tijd**afhankelijke requirements opstellen.
  - specificeren van periodetijden.
  - specificeren van deadlines.
- Verdelen van de beschikbare executie**tijd**.
  - voldoen aan de tijdafhankelijke requirements = scheduling (zie hoofdstuk 13)

THRijswijk

Tijd

Hoe snel is tijd?

- Elektronica is veel sneller dan de mechanische werkelijkheid.
- Voorbeeld: Auto botst tegen een muur. Hoeveel instructies kan de boordcomputer nog uitvoeren?
  - snelheid auto = 108 km/uur
  - kreukzone = 30 cm
  - processor freq = 200 MHz
  - gemiddeld 2 clockcycles / instructie

1.000.000

THRijswijk

RTC

real-time clock

- Geeft de “werkelijke tijd”.
- Java
  - System.currentTimeMillis()
    - Returns the current time in milliseconds. Note that while the unit of time of the return value is a millisecond, the granularity of the value depends on the underlying operating system and may be larger. Window granularity = 50 / 60 ms. QNX granularity = 1 ms.
  - class Date
- POSIX
  - int clock_gettime(clockid_t clock_id, struct timespec *res)
    - clock_id CLOCK_REALTIME is verplicht
  - int clock_getres(..)
    - granualarity POSIX max 20 ms QNX 1 ms
  - stuct tm
- Details zie boek 12.2.3 en 12.2.4.

THRijswijk

Slapen

- POSIX
  - sleep (granularity 1 sec)
  - nanosleep (granularity van CLOCK_REALTIME)
- Java
  - sleep method van class Thread (granularity van System)
- garandeert een minimale slaaptijd. Zie figuur 12.1 (p. 422)

THRijswijk

Time-out

- POSIX
  - semaphore:
    - int sem_**timed**wait(sem_t* sem, const struct timespec* abs_timeout);
  - mutex:
    - int pthread_mutex_**timed**lock(..., const struct timespec* abs_timeout);
  - conditionele variabele:
    - int pthread_cond_**timed**wait(..., const struct timespec* abs_timeout);
  - message queue:
    - mq_**timed**receive(..., const struct timespec* abs_timeout);
    - int mq_**timed**send(..., const struct timespec* abs_timeout);
  - Al deze ...**timed**... functies geven ETIMEDOUT terug bij een time-out
- Java
  - Object synchronisatie:
    - void wait(long timeout)

THRijswijk

Time-out

```
#include <pthreads.h>

int teller;
pthread_mutex_t tm =
   PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t tc =
   PTHREAD_COND_INITIALIZER;
...
   struct timespec ts;
   int r=0;
   pthread_mutex_lock(&tm);
   clock_gettime(CLOCK_REALTIME, &ts);
   ts.tv_sec += 5;
   while (teller<10 && r==0)
      r=pthread_cond_timedwait(&tc, &tm, &ts);
   if (r!=0) {
      if (r==ETIMEDOUT) /* time-out */
      else /* error */
   }
   else {
      // ...
      teller-=10;
   }
   pthread_mutex_unlock(&tm);
```

Handig om een absolute time te gebruiken!

THRijswijk

Time-out

```
public class Teller {
   private int teller=0;
   public synchronized void verwerkTienTallen()
         throws InterruptedException {
      long ts=System.currentTimeMillis();
      long dt=5000;
      long te=ts+dt;
      while (teller<10 && dt>0) {
         wait(dt);
         dt=te-System.currentTimeMillis();
      }
      if (teller<10) /*time-out */;
      else teller-=10;
      System.out.println(
         "teller is verlaagd en is nu "+teller);
   }
   public synchronized void verhoog(int n) {
      teller+=n;
      notify();
   }
}
```

THRijswijk

Time requirements

Temporal Scopes

- deadline
  - TS moet eindigen voor deadline. (hard, soft, firm)
- minimum delay
  - TS mag pas starten na min delay.
- maximum delay
  - TS moet starten na max delay.
- maximum execution time
  - TS mag niet langer runnen.
- maximum elapse time
  - TS mag niet langer duren.
- Soorten TS:
  - periodic (vaste tijd tussen events)
  - aperiodic (random events)
    - Geen worst-case analyse mogelijk.
  - sporadic (min tijd tussen events)

THRijswijk

Drift

- De extra vertaging bij sleep wordt local drift genoemd. Deze drift kan niet voorkomen worden.
- Je kunt wel voorkomen dat een cummulatieve drift ontstaat doordat local drifts bij elkaar worden opgeteld.

```
while (1) {
   actie();
   sleep(5); // slaap minstens 5 seconden
}
```

cummulatieve drift!

```
while (1) {
   actie();
   sigwait(&set, &signum); // slaap tot signal
}
```

local drift!

Met een timer (zie program 12.12) kan een periodiek signal (zie 10.6) worden opgewekt.

THRijswijk

Timing error detectie

- deadline overrun detectie.
  - zet een POSIX timer (blz. 451) die een signal (zie 10.6) geeft als de deadline verstrijkt = watchdog timer.
  - zie p. 450 t/m 452
- maximum execution time overrun detectie
  - POSIX definieert clocks: CLOCK_PROCESS_CPUTIME_ID en CLOCK_THREAD_CPUTIME_ID
  - zie p. 455 en 456

THRijswijk