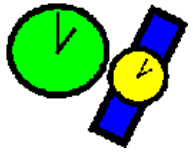
Real-Time Software (RTSOF)

EVMINX9 Week 4

DE HAAGSE HOGESCHOOL

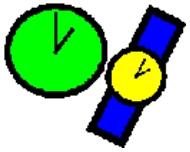
Real-time faciliteiten



Wat willen we met tijd in een RT systeem?

- Gebruik van de tijd.
  - Tijd(sduur) meten.
  - (Tot een) bepaalde tijd slapen.
  - Beperkte tijd wachten = time-outs.
- Tijdafhankelijke requirements opstellen.
  - Specificeren van periodetijden.
  - Specificeren van deadlines.
  - Verdelen van de beschikbare executietijd.
- Voldoen aan de tijdafhankelijke requirements = scheduling (zie hoofdstuk 13).

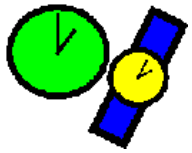
Hoe snel is tijd?



- 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

RTC real-time clock



- POSIX (niet nauwkeurig, granularity = 1 s)
  - time_t, stuct tm, time() and localtime() Zie PROS2: http://bd.thrijswijk.nl/pros2/time.htm

IEEE

- POSIX optional REALTIME (nauwkeuriger)
  - int clock_gettime(clockid_t clock_id, struct timespec *res)
    - clock_id CLOCK_REALTIME is verplicht
  - int clock_getres(..)
    - granularity POSIX max 20 ms,
      QNX instelbaar (default 1 ms, minimum 10 µs)

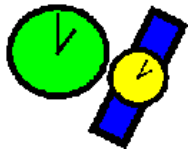
QNX

- C++ Boost (niet nauwkeurig, granularity = 1 s)
  - Date_Time

boost C++ LIBRARIES

DE HAAGSE HOGESCHOOL

Slapen



- POSIX (niet nauwkeurig, granularity = 1 s)
  - unsigned sleep(unsigned)

- POSIX optional REALTIME (nauwkeuriger)
  - int nanosleep(struct timespec* rqtp,
    struct timespec* rmtp);
    - Granularity van CLOCK_REALTIME
      QNX instelbaar (default = 1 ms, minimum = 10 µs)
- C++ Boost (granularity = ?)
  - boost::this_thread::sleep()

POSIX timeout

- Semaphore
  - int sem_timedwait(sem_t* sem, const struct timespec* abs_time);
- Mutex
  - int pthread_mutex_timedlock(..., const struct timespec* abs_time);
- Conditionele variabele
  - int pthread_cond_timedwait(..., const struct timespec* abs_time);
- Message queue
  - ssize_t mq_timedreceive(..., const struct timespec* abs_time);
  - int mq_timedsend(..., const struct timespec* abs_time);

Al deze ...timed... functies geven ETIMEDOUT terug bij een timeout

POSIX timeout


```
int ei_teller;
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t c = PTHREAD_COND_INITIALIZER;
...
    struct timespec ts; int r=0;
    pthread_mutex_lock(&m);
    clock_gettime(CLOCK_REALTIME, &ts);
    ts.tv_sec += 5;
    while (ei_teller<12 && r==0)
        r=pthread_cond_timedwait(&c, &m, &ts);
    if (r!=0) {
        if (r==ETIMEDOUT) /* time-out */
        else /* error */
    }
                                    else {
                                        // ...
                                        ei_teller-=12;
                                    }
                                    pthread_mutex_unlock(&m);
                                ...
```

Handig om een absolute time te gebruiken!
Snap je waarom?

Boost timeout

- thread
  - bool timed_join(const system_time& abs_time);
- timed_mutex, recursive_timed_mutex
  - bool timed_lock(const system_time& abs_time);
- shared_mutex
  - bool timed_lock(const system_time& abs_time);
  - bool timed_lock_shared(const system_time& abs_time);
- condition_variable, condition_variable_any
  - bool timed_wait(… lock, system_time const& abs_time);

Al deze ...timed... memberfuncties geven false terug bij een timeout

Boost timeout

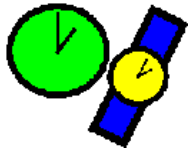

```
...
int ei_teller(0);

mutex m;
condition_variable c;

//  consumer thread
...
   {
      unique_lock<mutex> u(m);
      if (c.timed_wait(u, posix_time::seconds(5), []() {
          return ei_teller>=12;
       }))
         ei_teller-=12;
      else
         cout<<"Schiet eens op!"<<endl;
   }
...
```

```
//  producer thread
...
   {
      lock_guard<mutex> g(m);
      ei_teller+=n;
   }
   c.notify_one();
...
```

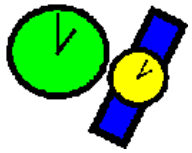
Time requirements



- Deadline → taak moet klaar voor deadline (hard, soft, firm)
- Minimum delay → taak mag pas starten na min delay
- Maximum delay → taak moet starten na max delay
- Maximum execution time → taak mag niet langer runnen
- Maximum elapse time → taak mag niet langer duren

- Soorten taken
  - Periodic (vaste tijd tussen events)
  - Aperiodic (random events)
    - Geen worst-case analyse mogelijk.
  - Sporadic (min tijd tussen events is gespecificeerd)

Drift



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

cumulatieve drift

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

lokale drift

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

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

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