Real-Time Systems (RTSYST)

Week 5

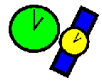
DE HAAGSE HOGESCHOOL

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.
- Voldoen aan de tijdafhankelijke requirements = scheduling (zie hoofdstuk 11).

Hoe snel is tijd?

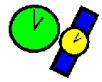


- Elektronica is veel sneller dan de mechanische werkelijkheid.
- Voorbeeld:
  Auto botst tegen een muur. Hoeveel instructies kan de boordcomputer nog uitvoeren?
  - snelheid auto = 108 km/uur
  - kreukzone = 30 cm
  - processor freq = 200 MHz
  - gemiddeld 2 clockcycles / instructie

1.000.000

DE HAAGSE HOGESCHOOL 137

RTC real-time clock



- POSIX (niet nauwkeurig, granularity = 1 s)
  - `time_t`, `stuct tm`, `time()` and `localtime()`
    Zie MICPRG: http://bd.eduweb.hhs.nl/micprg/time.htm
- POSIX optional REALTIME (nauwkeuriger)
  - `int clock_gettime(clockid_t clock_id, struct timespec *tp)`
    - `clock_id CLOCK_REALTIME` is verplicht
  - `int clock_getres(clockid_t clock_id, struct timespec *res)`
    - granularity POSIX max 20 ms,
      QNX instelbaar (default 1 ms, minimum 10 µs)
- C++11 (granularity = ?)
  - `chrono::system_clock`, `chrono::high_resolution_clock`

138

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++11 (granularity = ?)
  - `std::this_thread::sleep_for(… rel_time)`
  - `std::this_thread::sleep_until(… abs_time)`

139

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

DE HAAGSE HOGESCHOOL 140

POSIX timeout


```
int ei_teller = 0;
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)
        else /* error */
    }
                                    else {
                                        ei_teller -= 12;
                                    }
                                    pthread_mutex_unlock(&m);
                                //  ...
```

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

C++ timeout

- `timed_mutex`, `recursive_timed_mutex`
  - `bool try_lock_until(const chrono::time_point<C, D>& abs_time);`
  - `bool try_lock_for(const chrono::duration<R, P>& rel_time);`
- `condition_variable`
  - `cv_status wait_until(unique_lock<mutex>& lock, const chrono::time_point<C, D>& abs_time);`
  - `bool wait_until(unique_lock<mutex>& lock, const chrono::time_point<C, D>& abs_time, … p);`
  - `cv_status wait_for(unique_lock<mutex>& lock, const chrono::duration<R, P>& rel_time);`
  - `bool wait_for(unique_lock<mutex>& lock, const chrono::duration<R, P>& rel_time, … p);`

Memberfuncties die een `bool` teruggeven, geven `false` terug bij een timeout.
Memberfuncties die `cv_status` teruggeven, geven `cv_status::timeout` terug bij een timeout.

142

C++11 timeout


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

mutex m;
condition_variable c;

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

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

.43

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)

DE HAAGSE HOGESCHOOL 144

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 10.4.2) kan een periodiek signal (zie 7.5.1) worden opgewekt.

Timing error detectie

- Deadline overrun detectie
  - Zet een POSIX timer (10.4.2) die een signal (zie 7.5.1) geeft als de deadline verstrijkt = watchdog timer
  - Zie 13.2 en 13.2.3
- Maximum execution time overrun detectie
  - POSIX definieert clocks: `CLOCK_PROCESS_CPUTIME_ID` en `CLOCK_THREAD_CPUTIME_ID`
  - Zie 13.3 en 13.3.1

