

Concurrent systems modelling

TCTI-V2CSM1-16

Les 1, introductie

Vandaag

- ➔ Over de cursus.
- Opzet van de cursus.
- Concurrency.
- Problemen met concurrency.



Waar gaat deze cursus over?

- **Software Architectuur**
- **Parallel Programmeren**

Software Architectuur (Modelling)

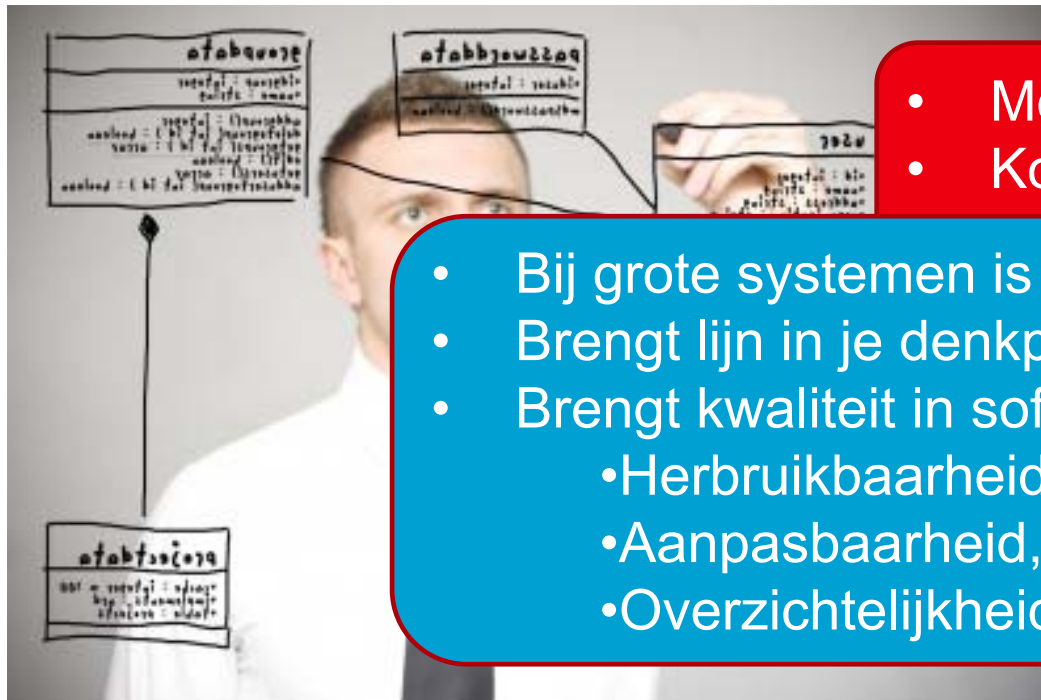
- **Modellen**
- **Mooi werk + goed salaris**

Parallel Programmeren

- Gelijktijdige deelprogrammas
- Overzichtelijk
- Krachtig
- Valkuilen

Meer over modelling

Een systeem uitdenken voor je gaat programmeren.



- Moeilijk
- Kost tijd.

- Bij grote systemen is een bouwplan handig.
- Brengt lijn in je denkproces.
- Brengt kwaliteit in software
 - Herbruikbaarheid,
 - Aanpasbaarheid,
 - Overzichtelijkheid dus minder kans op fouten.

Leerdoel van TCTI-V2CSM1-16.

Het ontwerpen van **embedded systemen** waarin verschillende **taken tegelijk** dienen te worden uitgevoerd en waarvoor **real time eisen** bestaan.

Embedded systems

Systemen die een processor gebruiken, maar waarvan de primaire functie niet die van een standaard computer is.

- B.v. een fietscomputer is een embedded systeem; het bevat een processor die niet is bedoeld als tekstverwerker of spreadsheet.

Real time systems.

Systemen waarin de tijd waarop de uitvoer wordt geproduceerd van essentieel belang is. Uitvoer die te laat of te vroeg komt is incorrect, ook al is de geleverde data correct.

- **Hard** - harde deadline voor de tijd
 - Reageren op overschrijding temperatuur in chemische fabriek.
- **Soft** - tijd mag afwijken maar het is niet wenselijk.
 - Videoplayer. Als het beeld hapert is dat niet leuk, maar het is geen ramp.

Concurrent systems

Systemen die meer dan een taak gelijktijdig (concurrent) uitvoeren.

- Een videospeler toont videobeelden en verwerkt tegelijk de knoppen die een gebruiker indrukt. Bovendien worden op de achtergrond videobeelden ontvangen via het netwerk en in een buffer geplaatst.

Realtime concurrent embedded system



WAT KAN HIER
MISGAAN ALS WE
SLECHTS ÉÉN
DING PER KEER
KUNNEN
AFHANDELEN?

Agenda

- Over de cursus.
- ➔ Opzet van de cursus (loop even langs canvas).
- Concurrency.
- Problemen met concurrency.

Belangrijk



- Eerst **oefenopdrachten** tijdens het college en de **huiswerkopdracht** maken.

Daarna pas aan je ontwerpdocument met de nieuwe deelopdracht toevoegen en uploaden.

- Wat je upload wordt **beoordeeld**
- Kan het zijn dat een docent **aanpassingen** vereist.
- Voldoende beoordeelde deelopdrachten behouden ook na aanpassing het **originele cijfer**.
- Onvoldoende beoordeelde deelopdrachten vereisen altijd aanpassing. Na voldoende aanpassing kan het cijfer ervoor **maximaal een 5.5** worden.



Bijhouden = King

- Lange feedback cycli
- Quid pro quo

Studiemateriaal

Studiemateriaal

- [Design Like a Robot!.zip](#)
- [Reader TCTI-V2CSM1-16.pdf](#)
- Al het genoemde in het programma op Canvas (voornamelijk sheets)

Aanbevolen naslagwerk

Hassan Gomaa (2016), *Real-Time Software Design for Embedded Systems*. Cambridge Press, May 2016.

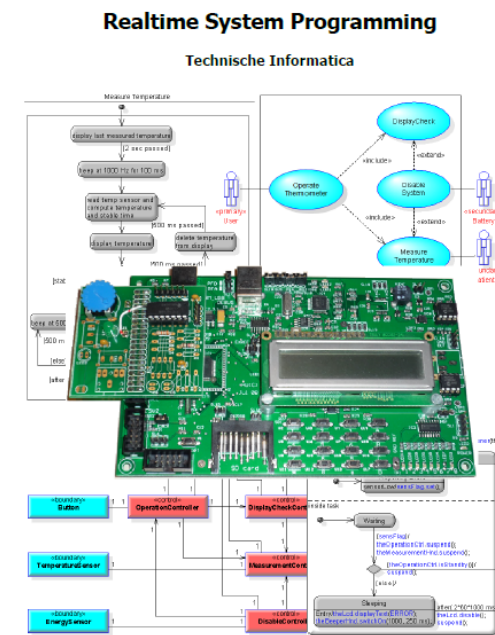
ISBN: 9781107041097

~~Online te lezen op~~
~~www.mediatheek.hu.nl~~.



Meer aanbevolen naslagwerk

Wensink en Van Ooijen
(2013). *Reader real-time
system programming.*
Hogeschool Utrecht, februari
2013.



Instituut voor Informatie- en Communicatie Technologie

M.Wensink / W.v.Ooijen, cursus 2012-2013
2013-02-05

Agenda

- Over de cursus.
- Opzet van de cursus.
- ➔ Concurrency.
- Problemen met concurrency.

Concurrent systems.

Systemen die meer dan een taak gelijktijdig (concurrent) uitvoeren.

- Een videospeler toont videobeelden en verwerkt tegelijk de knoppen die een gebruiker indrukt. Bovendien worden op de achtergrond videobeelden ontvangen via het netwerk en in een buffer geplaatst.

Waarom concurrency?

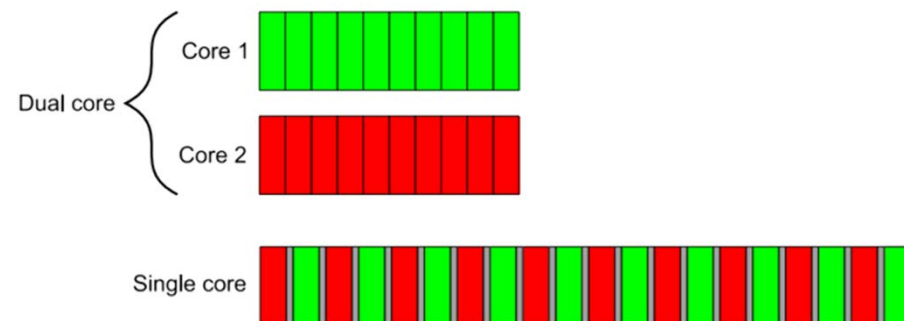
- Helder concept. In werkelijk gebeuren ook dingen tegelijk.
- Helpt om deadlines te halen in realtime systemen.
- Als er meerdere kernen kan je die allemaal gebruiken.

Hoe concurrency?

Concurrency:

Meer dan één taak tegelijkertijd uitvoeren

- Meerdere processors/cores
- Task switching



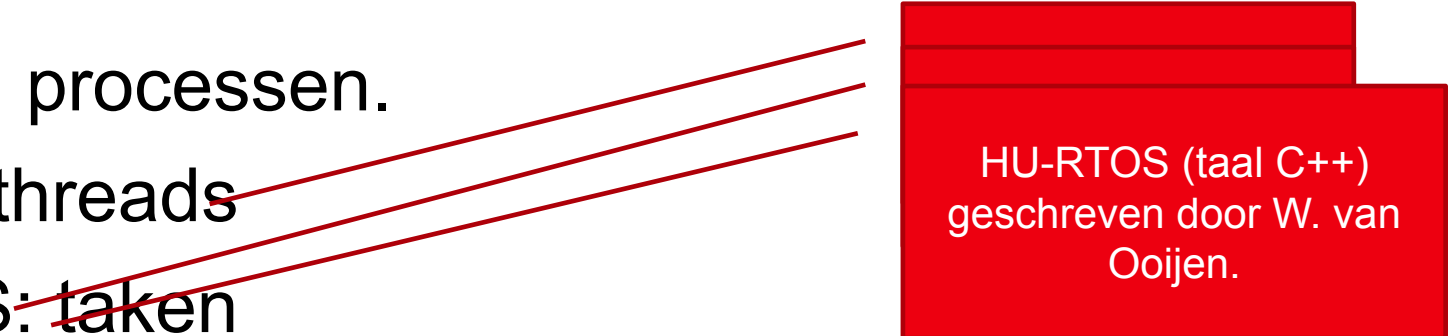
Anthony Williams (2012), *C++ Concurrency in Action - Practical Multithreading*. Manning, 2012, p.3.

Hoe concurrency?

Linux: processen.

C++: threads

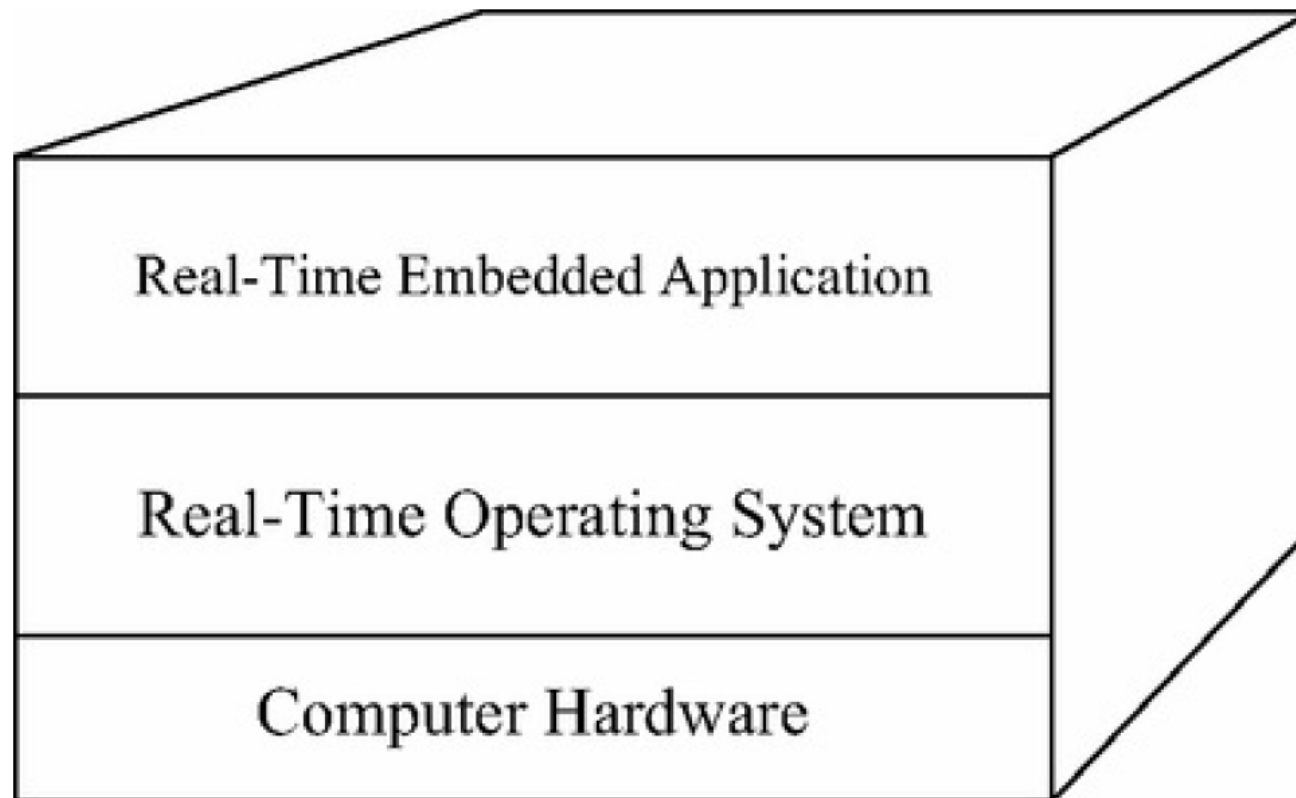
RTOS: ~~taken~~



HU-RTOS (taal C++)
geschreven door W. van
Ooijen.

- Een RTOS bevat een scheduler die verschillende taken kan schedulen.
- In een RTOS dienen taken verschillende prioriteiten te kunnen hebben.

Architectuur van realtime system.



Scheduling

- Preëemptive scheduling.
 - De scheduler bepaalt wanneer een taak zijn beurt moet afstaan aan de volgende taak.
- Coöperative scheduling.
 - De taak staat zelf zijn beurt af en dan kan de volgende taak verder.

Scheduling

Task running

Preemptive scheduler

Task ready



Buffering: Producer Consumer.

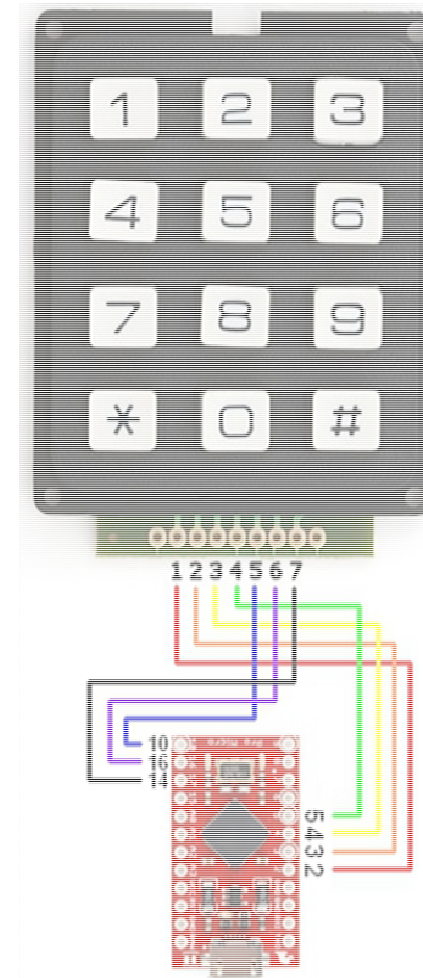


- Twee taken kunnen verschillende snelheid hebben.
- Om te zorgen dat de ene taak niet op de ander hoeft te wachten is een buffer nodig.
- Bij het afwassen is het bordenrekje de buffer.

Producers en consumers in embedded systemen.



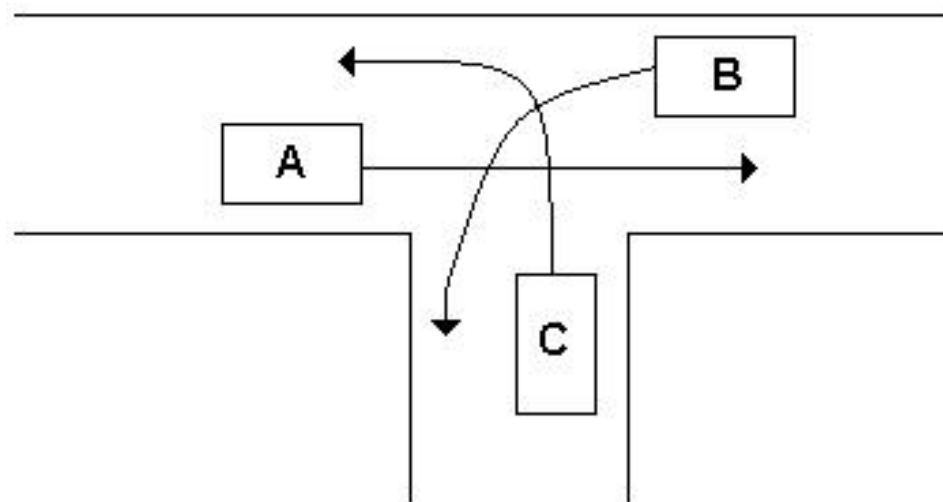
- Bijvoorbeeld toetsenbordtaak en verwerктаak.
- De gebruiker toetst sneller dan er verwerkt kan worden.
- Er is nu een buffer nodig de de toetsaanslagen onthoudt tot de verwerктаak ze kan verwerken.



Agenda

- Over de cursus.
- Opzet van de cursus.
- Concurrency.
- ➔ Problemen met concurrency.

Problemen met concurrency (deadlock)



Wie gaat eerst?

Problemen met concurrency (livelock)

Sommige taken komen niet aan de beurt:

- ☐ Taken met hoge prioriteit gebruiken alle processortijd.
- ☐ Bij coöperative scheduling: een lang durende taak staat de beurt niet af.

Problemen met concurrency (race condition)

Verschillende taken bewerken dezelfde gegevens.

Deze gegevens kunnen corrupt raken.

(bijvoorbeeld twee personen boeken dezelfde hotelkamer)

Demo threads

```
#include <iostream>
#include <string>
#include <thread>

void call_from_thread(std::string threadname) {
    for (int i = 0; i <= 100; i++) { std::cout << threadname; }
}

int main() {
    std::thread t1(call_from_thread, "a");
    std::thread t2(call_from_thread, "b");
    std::thread t3(call_from_thread, "c");

    t1.join();
    t2.join();
    t3.join();

    std::cout << std::endl;
    return 0;
}
```

Called from
thread

Launch 3 concurrent
threads

Called when all
threads have
finished

Demo threads



```
C:\Windows\system32\cmd.exe

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccc
Press any key to continue . . .
```



Demo race condition

```
#include <iostream>
#include <thread>

using namespace std;

int teller = 0;
const int AANTAL = 100000;

void telop() {
    for (int i = 0; i < AANTAL; i++) {
        teller++;
    }
}
```

```
int main(int argc, char **argv) {
    thread t1(telop);
    thread t2(telop);
    thread t3(telop);
    thread t4(telop);
    t1.join();
    t2.join();
    t3.join();
    t4.join();
    cout << "'\nteller = " << teller;
    cout << " (" << 4 * AANTAL << " expected)";
    cout << endl;
    return 0;
}
```

34³⁶

Demo race condition

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background and white text. The text displayed is:

```
teller = 303487 (400000 expected)
Press any key to continue . . .
```

The window is partially obscured by another window on top of it, which is also titled "C:\Windows\system32\cmd.exe". The top window shows the text "tel" and "Pre" on separate lines.

Demo mutex

```
#include <iostream>
#include <thread>
#include <mutex>
using namespace std;

int teller = 0;
const int AANTAL = 100000;
mutex mtx;

void telop() {
    for (int i = 0; i < AANTAL; i++) {
        mtx.lock();
        teller++;
        mtx.unlock();
    }
}
```

```
int main(int argc, char **argv) {
    thread t1(telop);
    thread t2(telop);
    thread t3(telop);
    thread t4(telop);
    t1.join();
    t2.join();
    t3.join();
    t4.join();
    cout << "\nteller = " << teller
    cout << " (" << 4 * AANTAL << " expected)"
    cout << endl;
    return 0;
}
```



Demo mutex



```
C:\Windows\system32\cmd.exe  
teller = 400000 (400000 expected)  
Press any key to continue . . .
```

Demo deadlock



```
int teller1 = 0;
int teller2 = 0;

const int AANTAL = 100000;

mutex mtx_teller1;
mutex mtx_teller2;

void doe_iets_12() {
    for (int i = 0; i < AANTAL; i++) {
        mtx_teller1.lock();
        mtx_teller2.lock();
        teller2 = teller1+1;
        teller1 = teller2;
        mtx_teller1.unlock();
        mtx_teller2.unlock();
    }
}
```

```
void doe_iets_21() {
    for (int i = 0; i < AANTAL; i++) {
        mtx_teller2.lock();
        mtx_teller1.lock();
        teller2 = teller1+1;
        teller1 = teller2;
        mtx_teller2.unlock();
        mtx_teller1.unlock();
    }
}

int main(int argc, char **argv) {
    thread t1(doe_iets_12);
    thread t2(doe_iets_21);
    t1.join();
    t2.join();
    cout << "we are lucky if we have not crashed" <<
teller
    cout << endl;

    return 0;
}
```

Opdracht: Dansend ASCII - poppetje



- Thread1: cout ascii-popje op basis toestandsvariabelen voor wenkbrauwen, ogen, neus, mond en lichaam:

```
if(wenkbrauw==boos){cout << "\\ /";}else{cout<< "w w";}
if(ogen==gesloten){cout << "- -";}else{cout<< "O O";}
```
- Thread2: verandert periodiek ogen-toestandsvariabele (knipperen).

```
if(ogen==dicht){ogen=open;}else{ogen=dicht;}
```
- Thread3: verandert periodiek wenkbrouwen-toestandsvariabele. Bijvoorbeeld:

```
if(wenkbrauw==boos){wenkbrauw=neutraal;}else{wenkbrauw=boos;}
```
- Thread4: verandert periodiek die van de mond (praten), maar is ook afhankelijk van de toestand van de wenkbrauwen.

```
if(wenkbrauw==boos){mond=boos;}else{..kies wat anders voor mond}
```
- Thread5: ,, ,, van het lichaam.
- Bescherm toegang tot de toestandsvariabelen (zowel lezen als schrijven) elk met een eigen mutex.
- Tip: gebruik visual studio of visual code. Kies: "Console Application project".



Nabespreking

40³⁶

Samenvatting huilen voorkomen met parallel programmeren



- Gebruik synchronisatiemechanismen (zoals mutex) voor toegang tot gedeelde bronnen. [om races te voorkomen]
- Bij gebruik van mutexen – lock ze altijd in dezelfde volgorde. [om deadlocks te voorkomen]
- Lock gesharede resources zo kort mogelijk [om livelocks te voorkomen].
(NB: dat laatste geldt dus zowel voor cooperative als voor preemptive multitasking)



Terugblik.

RTOS

Concurrent Tasks

Deadlock

Scheduler

Realtime systems

*Preemptive multitasking vs
Coöperative multitasking*

Task switching

42³⁶