Real-Time Systems (RTSYST)

Week 1

DE HAAGSE HOGESCHOOL

Real-Time Systems (RTSYST)

- Onderwerpen:
  - Concurrent programming (threads).
  - Real-Time OS (VxWorks, **QNX**, FreeRTOS).
  - Concurrent programming in C en C++.
  - Synchronisation and Communication.
  - Real-Time faciliteiten (clocks, timeouts).
  - Scheduling.
- Werkvormen:
  - 14 lessen theorie + 7 lessen begeleid practicum.
  - 5 uur/week zelfstudie (inclusief onbegeleid practicum).

DE HAAGSE HOGESCHOOL 2

Leermiddelen

- Boeken
  - Real-Time Systems and Programming Languages (Fourth Edition), Alan Burns and Andy Wellings, ISBN: 9780321417459
    Hoofdstuk 1, 4 t/m 6, 9 en 11.
  - QNX Neutino 2, Robert Krten (kun je bij mij lenen)
- Blackboard en http://bd.eduweb.hhs.nl/rtsyst/
  - Studiewijzer met uitgebreide planning
  - Practicumopdrachten + uitgebreide practicumhandleiding
  - Sourcecode van alle voorbeelden
  - Sheets
  - Links

DE HAAGSE HOGESCHOOL 3

Real-Time Systeem

- Definitie(s):
  - Systeem waarvan de reactietijd op een onvoorspelbare inputverandering voorspelbaar is.
  - Systeem waarbij de uitvoer niet alleen correct moet zijn maar ook op het juiste moment.

DE HAAGSE HOGESCHOOL 4

Indeling Real-Time Systemen

- Hard real-time
  - Missen van een deadline is fataal.
- Soft real-time
  - Missen van een deadline is ongewenst.
- Interactief (niet real-time)
  - Er zijn geen expliciete deadlines maar wachten is wel irritant.

DE HAAGSE HOGESCHOOL 5

Voorbeelden Real-Time Systeem

- Procesbesturing (meet en regeltechniek)
- Productie besturingssysteem (industriële automatisering)
- Embedded systemen
  - ABS (Anti-Blokeer-Systeem)
  - Pacemaker
  - Besturing kruisraket
  - Kopieer apparaat
  - DVD recorder

Karakteristieken Real-Time Systeem

- Groot en complex (niet altijd)
  - Onderhoudbaar: uitbreidbaar, aanpasbaar en herbruikbaar
- Betrouwbaar en veilig
  - Intensive care apparatuur
  - Kerncentrale
  - Automatische piloot
- Concurrent gedrag
  - Multitasking, multiprocessor, distributed
  - RTOS of RTL moet dit ondersteunen
- Timing faciliteiten
  - Taak op bepaalde tijd starten, taak binnen bepaalde tijd afronden
  - RTOS of RTL moet dit ondersteunen
- Interactie met hardware

DE HAAGSE HOGESCHOOL 7

Concurrent programming

- Single processor system
  - Multitasking m.b.v. time sharing
- Multi processor system met gedeeld geheugen (SMP) of multi-core processor systeem
  - Parallel (true multitasking)
- Distributed system
  - Parallel
  - Verschillende systemen (elk met een eigen geheugen) verbonden met een netwerk

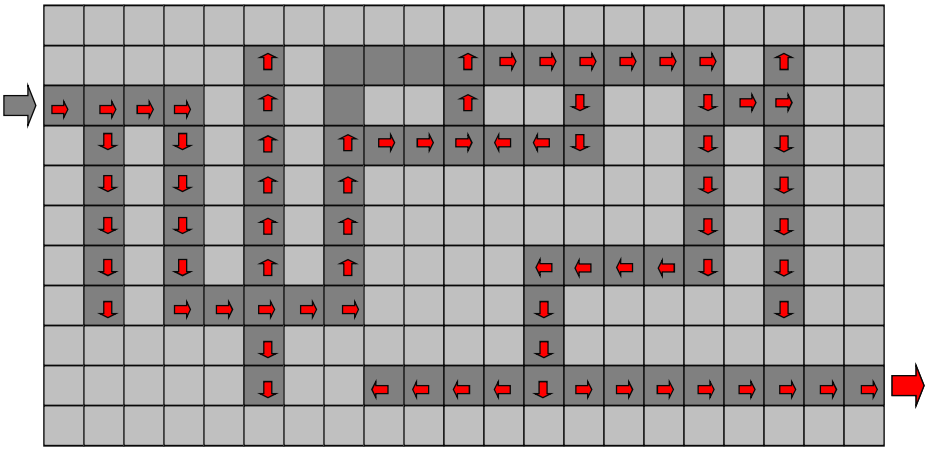
DE HAAGSE HOGESCHOOL 8

Why Concurrent programming

- Programma model komt overeen met de werkelijkheid
- Benutten van parallellisme in applicatie
  - Zoek je weg in een doolhof
  - Vergelijken van vingerafdrukken
- Processor beter benutten op single processor systeem

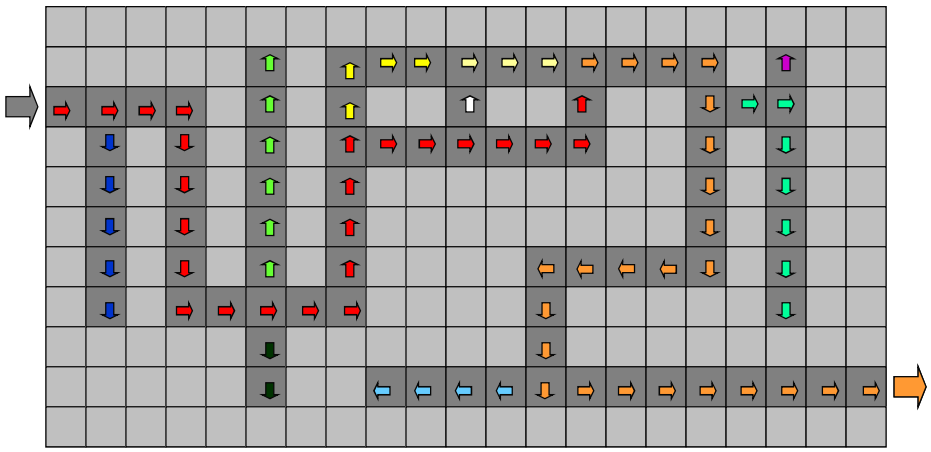
DE HAAGSE HOGESCHOOL 9

Sequential Maze Search



DE HAAGSE HOGESCHOOL 10

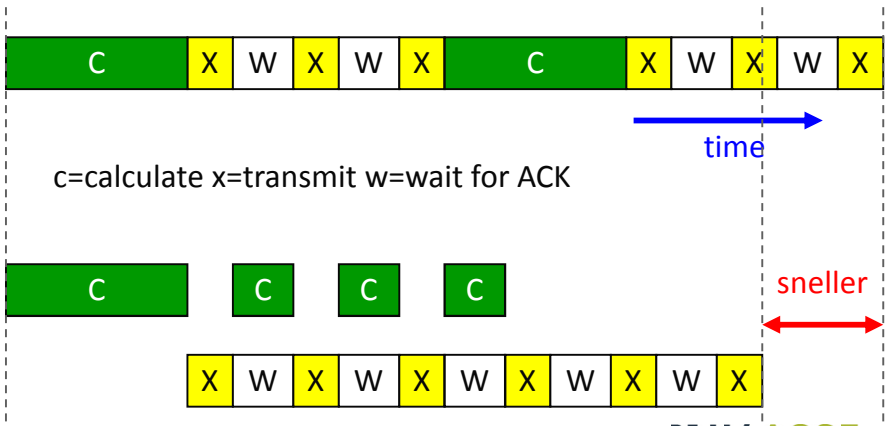
Concurrent Maze Search



DE HAAGSE HOGESCHOOL 11

Concurrent programming

- Processor beter benutten op single processor systeem



DE HAAGSE HOGESCHOOL 12

Beperking van parallellisme

- Amdahl's Law (boek p. 96)
  - De versnelling (speedup) van een program door het gebruik van meerdere parallelllle processoren (of cores) is begrensd door het deel van het programma dat sequentieel uitgevoerd moet worden.

$$S_N = \frac{1}{(1-P) + \frac{P}{N}}$$

  - N = aantal processoren
  - S_N = speedup met N processors (cores)
  - P = deel van het programma dat parallel uitgevoerd kan worden

- Voorbeeld:
  - Bereken de maximale speedup voor een programma waarvan 25% sequentieel uitgevoerd moet worden? (4)
  - Wat is de maximale speedup bij 2, 4, 8 en 16 cores? (1.60, 2.29, 2.91 en 3.37)

DE HAAGSE HOGESCHOOL 13

Process versus Thread

- Process
  - Eigen stack = veilig
  - Eigen (data) memory map
    - Eigen virtueel geheugen = veilig, communicatie = traag
  - Process switch is traag (zwaar)
    - Cache flush, MMU TLB flush

- Thread
  - Eigen stack = veilig
  - Gedeelde (data) memory map binnen hetzelfde process
    - Gedeeld geheugen = onveilig
    - Communicatie = snel
  - Thread switch is snel (licht) binnen hetzelfde process
    - Geen flushes

DE HAAGSE HOGESCHOOL 14

Process versus Thread

- Veel GPOSs (General Purpose Operating Systems) gebruiken:
  - Processes om verschillende applicaties van elkaar te scheiden
  - Threads om concurrency binnen applicatie mogelijk te maken
  - Voorbeelden: Windows en Linux
- Veel RTOSs (Real-Time Operating Systems) gebruiken:
  - Threads / Tasks om concurrency mogelijk te maken
  - Voorbeeld: FreeRTOS
- QNX gebruikt processes en threads
  - Wij behandelen alleen threads

DE HAAGSE HOGESCHOOL 15

QNX (POSIX compatible RTOS)

DE HAAGSE HOGESCHOOL 16

Huiswerk

- Bestudeer: Boek H1 t/m 1.3.
- Bestudeer: Artikel uit Embedded Computer Design: RTOS versus GPOS
- Achtergrondinformatie:
  - Artikel The Free Lunch Is Over
  - Artikel TLAs: QNX, RIM, ARM, CES, RPM, and MPH. An RTOS for the Automotively Inclined
  - Web seminar: Why do I need an RTOS anyway?

DE HAAGSE HOGESCHOOL 17

Concurrent programming

- Sequentiële programmeertaal (C of C++) + OS (Linux, Windows, QNX, FreeRTOS)
  - Portable als taal en OS portable zijn IEEE POSIX 1003
  - Meerdere talen combineren in 1 applicatie is mogelijk
- Concurrent programmeertaal (ADA, Java, C#, C++11, C11)
  - Beter leesbaar
  - Beter onderhoudbaar
  - Portable als taal portable is
- Concurrent framework (OpenMP, OpenCL)
  - Portable (ondersteund meerdere talen en OS-en)
- Middleware (RPC, RMI, CORBA)
  - Vereenvoudigt bouwen van distributed applicaties

DE HAAGSE HOGESCHOOL 18

Fundamentele vragen

- Hoe kun je processen / threads beheren?
  - Support in OS via API (= Application Programming Interface) of library
  - Support in programmeertaal
- Hoe kunnen processen / threads communiceren?
  - IPC = Inter Process Communication (term wordt ook voor communicatie tussen threads gebruikt)
- Hoe kun je processen / threads synchroniseren?
  - IPC zonder dataoverdracht.

DE HAAGSE HOGESCHOOL 19

Concurrent OOP

- Actieve objecten
  - Object heeft eigen thread of process.
  - Versturen zelf actief (spontaan) messages.
- Passieve objecten
  - Object heeft geen eigen thread of process.
  - Reageren op binnenkomende messages en kunnen als reactie:
    - Zelf message versturen.
    - Toestand van aanroepende thread of process veranderen (b.v. van running naar waiting).

DE HAAGSE HOGESCHOOL 20

Specificatie van concurrent taken

- Coroutine
  - Coöperatief (eerste Apple, win16)
- System call
  - UNIX, win32

```
pthread_t t;
pthread_create(&t, NULL, &func, NULL)
```

- Concurrent blok
  - Concurrent Pascal, High Performance Fortran

```
cobegin s1; s2; s3 coend;
```

- Expliciete declaratie
  - ADA, Java

```
task body NAME is
begin
…
end NAME;
```

1

Concurrent execution

IEEE POSIX 1003.1-2013

- `fork()` en `wait()`
- `posix_spawn()`
  - Combinatie van `fork()`, `exec()` en `wait()`
- `pthread_create()` en `pthread_join()`

Documentatie:

- IEEE Std 1003.1-2013 = The Open Group Base Specifications Issue 7
  http://pubs.opengroup.org/onlinepubs/9699919799/
- QNX documentation: http://www.qnx.com/developer/docs/

22

Pointers naar functies

- In C kun je een pointer naar een functie definiëren.
  - De waarde van de pointer is het beginadres (van de code) van de functie.

```
#include <stdio.h>

int kwadraat(int c) {
    return c * c;
}

int dubbel(int c) {
    return c + c;
}

int main(void) {
    int a = 7, b;
    int (*pnf)(int);
    pnf = &dubbel;
    b = (*pnf)(a);
```

pnf is een **pointer naar een functie** met een `int` als parameter en een `int` returnwaarde

Waarom haakjes?

Pointers naar functies

- In C kun je een pointer naar een functie definiëren.
  - De waarde van de pointer is het beginadres (van de code) van de functie.

```
#include <stdio.h>

int kwadraat(int c) {
    return c * c;
}

int dubbel(int c) {
    return c + c;
}

int main(void) {
    int a = 7, b;
    int (*pnf)(int);
    pnf = &dubbel;
    b = (*pnf)(a);
```

pnf **wijst naar** de functie dubbel (pnf wordt gelijk aan **het adres van** de functie dubbel)

Pointers naar functies

- In C kun je een pointer naar een functie definiëren.
  - De waarde van de pointer is het beginadres (van de code) van de functie.

```
#include <stdio.h>

int kwadraat(int c) {
    return c * c;
}

int dubbel(int c) {
    return c + c;
}

int main(void) {
    int a = 7, b;
    int (*pnf)(int);
    pnf = &dubbel;
    b = (*pnf)(a);
```

De functie **waar pnf naar wijst** wordt **aangeroepen** met de waarde van **a** als argument

Waarom haakjes?

DE HAAGSE HOGESCHOOL 25

Pointers naar functies

- Verkorte schrijfwijze.
  - Naam van een functie ←→ beginadres (van de code) van de functie.

```
#include <stdio.h>

int kwadraat(int c) {
    return c * c;
}

int dubbel(int c) {
    return c + c;
}

int main(void) {
    int a = 7, b;
    int (*pnf)(int);
    pnf = dubbel;
    b = pnf(a);
```

DE HAAGSE HOGESCHOOL 26

Pointers naar functies

- Wat is het nut?
  - Functie als parameter.

```
#include <stdio.h>
/* … */
void printTabel(int (*p)(int), int van, int tot, int stap) {
    int x;
    for (x = van; x < tot; x += stap) {
        printf("%10d %10d\n", x, (*p)(x));
    }
}
int main(void) {
    printf("De kwadraten van 1 t/m 10\n");
    printTabel(&kwadraat, 1, 11, 1);
    printf("De dubbelen van de drievouden van 0 t/m 30\n");
    printTabel(&dubbel, 0, 31, 3);
```


Uitvoer

```
De kwadraten van 1 t/m 10
         1          1
         2          4
         3          9
         4         16
         5         25
         6         36
         7         49
         8         64
         9         81
        10        100
De dubbelen van de drievouden van 0 t/m 30
         0          0
         3          6
         6         12
         9         18
        12         24
        15         30
        18         36
        21         42
        24         48
        27         54
        30         60
```


void*

- Een `void*` kan wijzen naar elk type.
- Als we de waarde willen ophalen waar een `void*` naar wijst dan moeten we de pointer casten naar het juiste type.

```
int main(void) {
    int i = 3;
    double d = 4.3;

    void* vp = &i;
    printf("%d\n", *(int*)vp);
    vp = &d;
    printf("%lf\n", *(double*)vp);

    return EXIT_SUCCESS;
}
```

DE HAAGSE HOGESCHOOL 29

pthread (1 van 3)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <pthread.h>

void check(int error) {
    if (error != 0) {
        fprintf(stderr, "Error: %s\n", strerror(error));
        exit(EXIT_FAILURE);
    }
}
```

DE HAAGSE HOGESCHOOL 30

pthread (2 van 3)

```
void* print1(void* par) {
    struct timespec ts = {0, 10000000};
    int i;
    for (i = 0; i < 10; i++) {
        nanosleep(&ts, NULL);
        printf("print1\n");
    }
    return NULL;
}
void* print2(void* par) {
    struct timespec ts = {0, 20000000};
    int i;
    for (i = 0; i < 10; i++) {
        nanosleep(&ts, NULL);
        printf("print2\n");
    }
    return NULL;
}
```

31

pthread (1 van 3)

```
int main(void) {
    pthread_t t1, t2;
    check( pthread_create(&t1, NULL, &print1, NULL) );
    check( pthread_create(&t2, NULL, &print2, NULL) );

    check( pthread_join(t1, NULL) );
    check( pthread_join(t2, NULL) );
    return EXIT_SUCCESS;
}
```

```
print1
print2
print1
print1
print2
print1
print1
print2
print1
print1
print2
print1
print1
print2
print1
print1
print2
print1
print2
print2
print2
print2
print2
```

pthread Alternatieve implementatie

```
typedef struct {
    char* msg;
    long ns;
} par_t;
```

SAFETY

Pas op: void*

```
void* print(void* p) {
    par_t* pp = p;
    struct timespec ts = {0, pp->ns};
    int i;
    for (i = 0; i < 10; i++) {
        nanosleep(&ts, NULL);
        printf(pp->msg);
    }
    return NULL;
}
int main(void) {
    pthread_t t1, t2;
    par_t p1 = {"print1\n", 10000000};
    par_t p2 = {"print2\n", 20000000};
    check( pthread_create(&t1, NULL, &print, &p1) );
    check( pthread_create(&t2, NULL, &print, &p2) );
//  ...
```

33

Command line argumenten

- Aan een programma kunnen command line argumenten worden doorgegeven.
  - Deze zijn in main beschikbaar via de parameters `argc` en `argv`
    - `int main(int argc, char* argv[]) { … }`
  - `argv[0]` is de naam van het programma.
  - `argc` geeft het aantal meegegeven parameters + 1
  - `argv[1]` is de eerste meegegeven parameter. Enz.

```
int main(int argc, char* argv[]) {
    int i;
    for (i = 0; i < argc; i++) {
        printf("%s\n", argv[i]);
    }
    return EXIT_SUCCESS;
}
```

```
$ a.exe dat is leuk
a.exe
dat
is
leuk
```

DE HAAGSE HOGESCHOOL

34