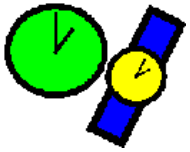
Real-Time Software (RTSOF)

EVMINX9 Week 3

DE HAAGSE HOGESCHOOL

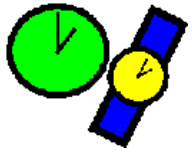
IPC inter process communication



- Shared variabele based (H8)

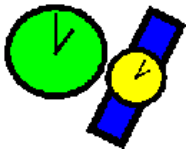
- Message based (H9)
  - Kan ook gebruikt worden in systemen zonder gedeeld geheugen (gedistribueerde systemen).
  - POSIX: message queue
  - QNX: Neutrino kernel is volledig message based. Zie H2 QNX boek.

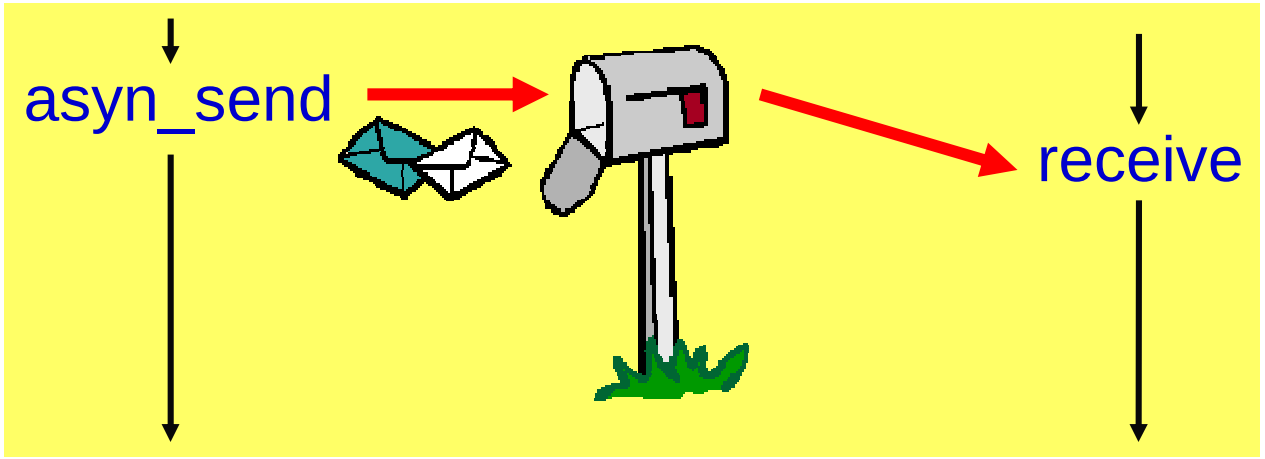
Messages Synchronisatie

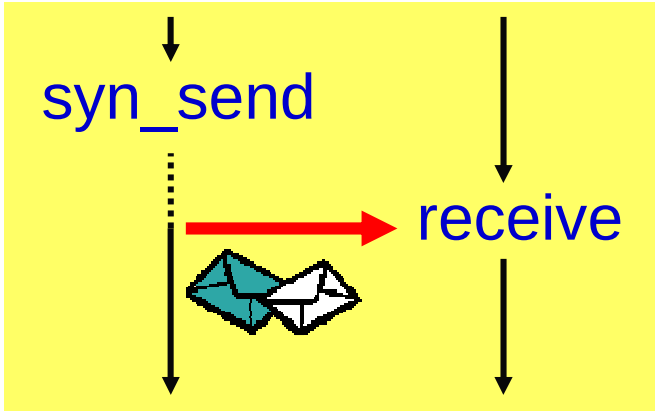


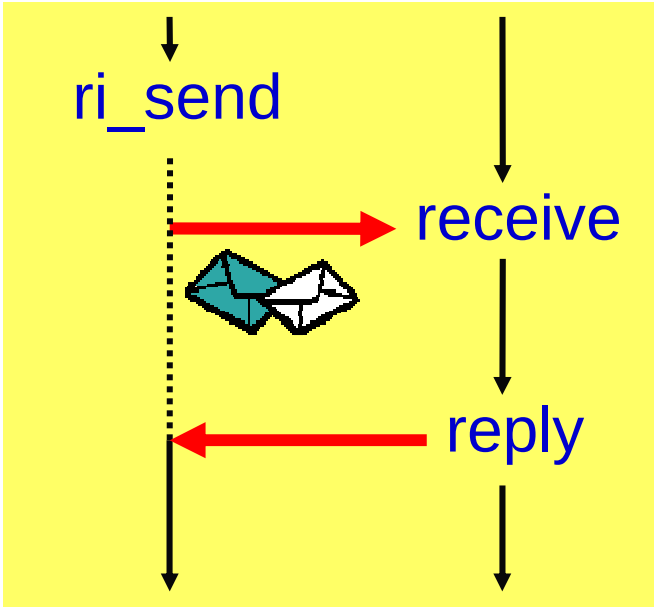
- Receive:
  - Wacht als er nog niet gezonden is.
- Send:
  - Asynchroon: wacht niet.
    - Buffer nodig, wat als buffer vol is?
    - Bijvoorbeeld: POSIX message queue.
  - Synchroon (rendevous): wacht op ontvangst.
    - Geen buffer nodig.
  - Remote invocation (extended rendevous): wacht op antwoord.
    - Geen buffer nodig.
    - Bijvoorbeeld: QNX messages.

Messages

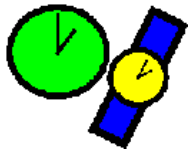








Messages



Synchroon met behulp van 2 asynchrone messages

Proces 1
```
asyn_send(mes)
receive(ack)
```

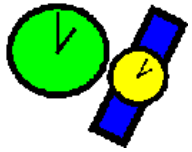
Proces 2
```
receive(mes)
async_send(ack)
```

Remote invocation met behulp van 4 asynchrone messages

Proces 1
```
asyn_send(mes)
receive(ack)
receive(reply)
async_send(ack)
```

Proces 2
```
receive(mes)
async_send(ack)
//... construct reply
async_send(reply)
receive(ack)
```

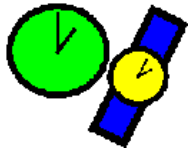
Asynchroon



- Voordelen:
  - Flexibeler.
- Nadelen:
  - Buffers nodig.
  - Complexer: Apparte message voor acknowledge en/of reply nodig.
  - Moeilijk om correctheid van een programma te bewijzen.

Asynchrone communicatie kan in een OS dat op synchrone messages is gebaseerd (QNX) worden gerealiseerd door expliciete buffer threads.

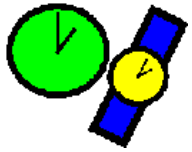
Messages Adressering



- Direct: sender geeft receiver proces (of thread) op.
- Indirect: sender geeft port, channel of mailbox op.

- Symetrisch: receiver geeft sender, port, channel of mailbox op.
- Asymetrisch: receiver geeft niets op.

Messages inhoud



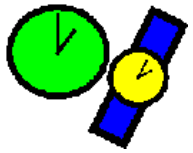
- Tussen threads:
  - Geen beperkingen.
- Tussen processen:
  - Geen pointers (elk proces heeft zijn eigen memory map).
- Tussen machines:
  - Geen pointers + mogelijk problemen met representatie:
    - Character codering.
    - Big-endian, Little-endian.

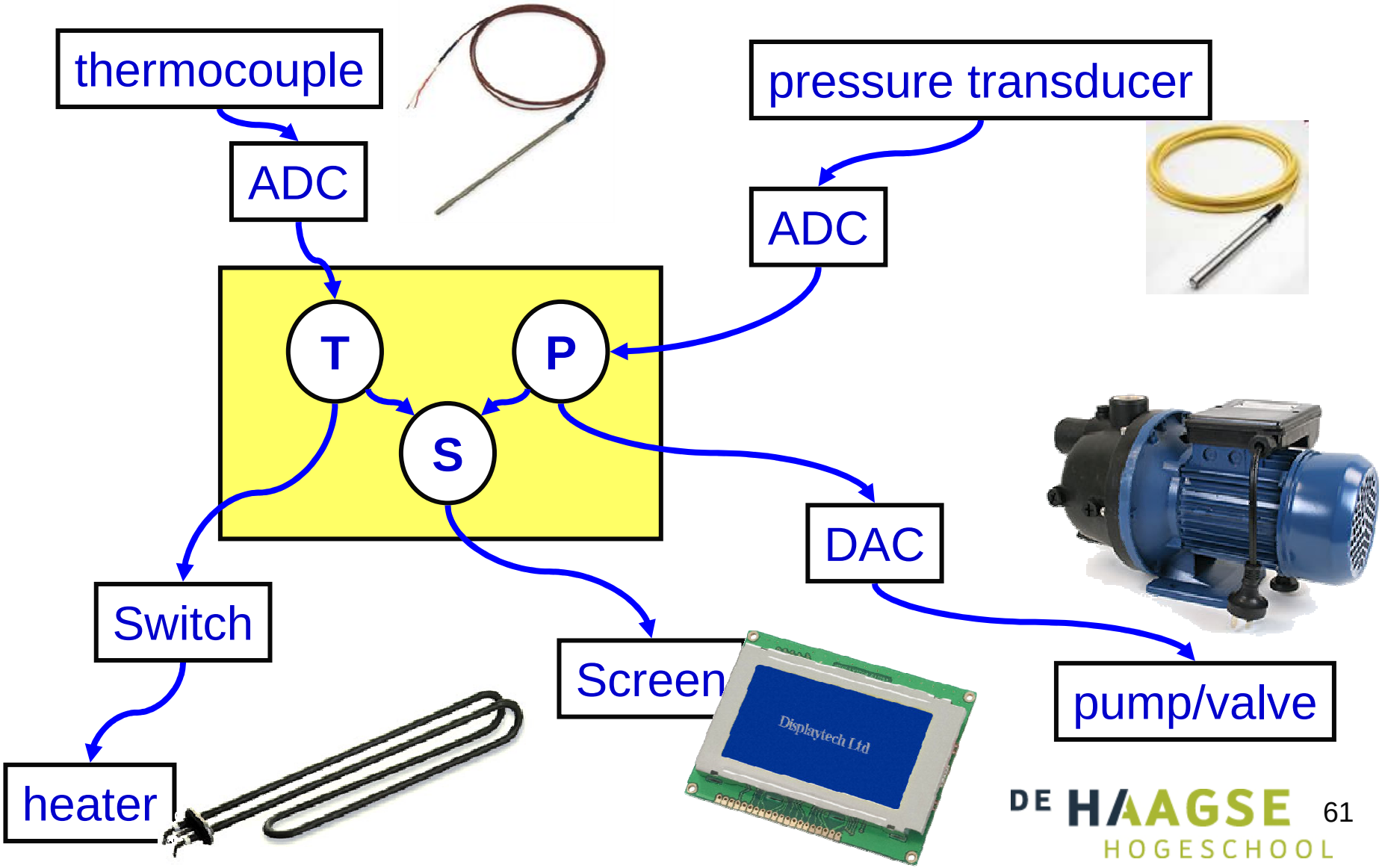
POSIX message queue

- Kenmerken:
  - Synchronisatie: asynchroon
  - Adressering: indirect en symetrisch
- Inhoud: geen beperkingen.
- Meerdere senders en receivers kunnen dezelfde mq gebruiken.
- Aan een message kan een prioriteit worden meegegeven.
- Bij creatie wordt o.a. opgegeven:
  - Naam
  - Max aantal messages
  - Max size message

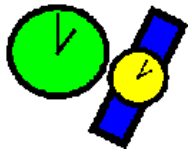
API:
mq_open (create en open)
mq_send, **mq_receive**
mq_close, mq_unlink (destroy)
mq_getattr, mq_setattr
mq_notify

Voorbeeld embedded system





Sequentieel

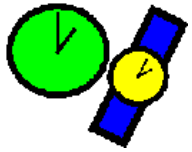


```
#include <stdio.h>
#include <stdlib.h>

double readTemp(void);
void writeSwitch(int i);
double readPres(void);
void writeDAC(double d);
int tempControl(double temp);
double presControl(double pres);

int main(void) {
    double temp, pres, dac;
    int switch_;
    while (1) {
        temp=readTemp();
        switch_=tempControl(temp);
        writeSwitch(switch_);
        pres=readPres();
        dac=presControl(pres);
        writeDAC(dac);
        printf(
            "%4.1lf, %4.1lf, %d, %5.1lf\n",
            temp, pres, switch_, dac);
    }
    return EXIT_SUCCESS;
}
```

Sequentieel problemen



- Sample rate van temperatuur en druk is gelijk.
  - Kan wel wat aan worden gedaan met tellers maar: Wat doe je als pressureControl langer duurt dan gewenste sample rate van temperatuur?
- Als readTemperature niet werkt (blijft pollen) dan loopt ook de drukregeling vast.

De temperatuurregeling en de drukregeling zijn twee afzonderlijke “processen”. Maar in het sequentiële programma zitten ze verweven!

Processen

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(void) {
    pid_t pid;
    pid=fork();
    if (pid==-1) {
        perror("fork");
        return EXIT_FAILURE;
    }
```

```
if (pid==0) { // this is the child
    double temp;
    int switch_;
    while (1) {
        temp=readTemp();
        printf("temperature = %4.1lf, ",
            temp);
        switch_=tempControl(temp);
        printf("switch = %d\n", switch_);
        writeSwitch(switch_);
        sleep(3);
    }
}
```

Zie volgende sheet…

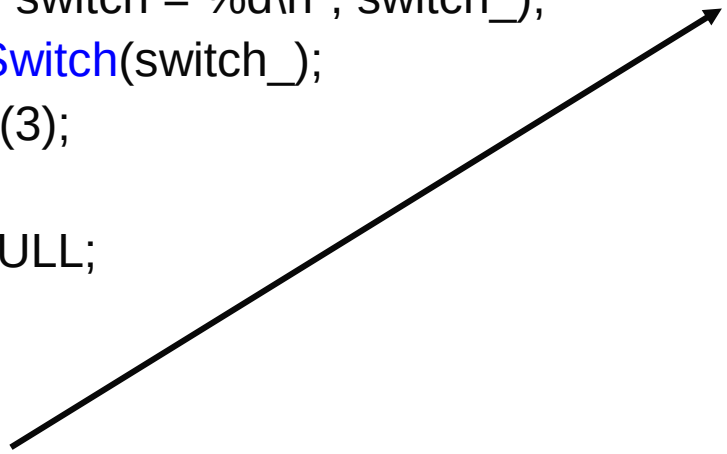
Processen


```
    else { // this is the parent
        double pres, dac;
        while (1) {
            pres=readPres();
            printf("pressure = %4.1lf", pres);
            dac=presControl(pres);
            printf(", DAC = %5.1lf\n", dac);
            writeDAC(dac);
            sleep(1);
        }
        wait(0);
    }
    return EXIT_SUCCESS;
}
```

```
temperature =  1.0, switch = 0
pressure = 18.4, DAC =  -8.4
pressure = 18.0, DAC =  -8.0
pressure = 17.6, DAC =  -7.6
temperature =  1.5, switch = 0
pressure = 17.2, DAC =  -7.2
pressure = 16.8, DAC =  -6.8
pressure = 16.4, DAC =  -6.4
```

Threads

```
void* tempThread(void* p) {
    double temp;
    int switch_;
    while (1) {
        temp=readTemp();
        printf("temperature = %4.1lf, ", temp);
        switch_=tempControl(temp);
        printf("switch = %d\n", switch_);
        writeSwitch(switch_);
        sleep(3);
    }
    return NULL;
}
```



Zie volgende sheet…

```
void* presThread(void* p) {
    double pres, dac;
    while (1) {
        pres=readPres();
        printf("pressure = %4.1lf", pres);
        dac=presControl(pres);
        printf(", DAC = %5.1lf\n", dac);
        writeDAC(dac);
        sleep(1);
    }
    return NULL;
}
```

Threads

```
#include <pthread.h>

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

int main(void) {
    pthread_t t1, t2;
    check( pthread_create(&t1, NULL, tempThread, NULL) );
    check( pthread_create(&t2, NULL, presThread, NULL) );
    check( pthread_join(t1, NULL) );
    check( pthread_join(t2, NULL) );
    return EXIT_SUCCESS;
}
```

```
pressure = 14.4, DAC =  -4.4
pressure = 14.0temperature =  3.0, , DAC =  -4.0
switch = 0
pressure = 13.6, DAC =  -3.6
```

Synchronized Threads


```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void* tempThread(void* p) {
    double temp;
    int switch_;
    while (1) {
        temp=readTemp();
        check( pthread_mutex_lock(&mutex) );
        printf("temperature = %4.1lf, ", temp);
        switch_=tempConvert(temp);
        printf("switch = %d\n", switch_);
        check( pthread_mutex_unlock(&mutex) );
        writeSwitch(switch_);
        sleep(3);
    }
    return NULL;
}
```

Synchronized Threads

```
class Temp {
private:
    double temp;
    bool switch_;
    boost::mutex& display;
    void read();
    void write();
    void control();
public:
    Temp(boost::mutex& m);
    void operator()();
};
```

```
class Pres {
private:
    double pres, dac;
    boost::mutex& display;
    void read();
    void write();
    void control();
public:
    Pres(boost::mutex& m);
    void operator()();
};
```

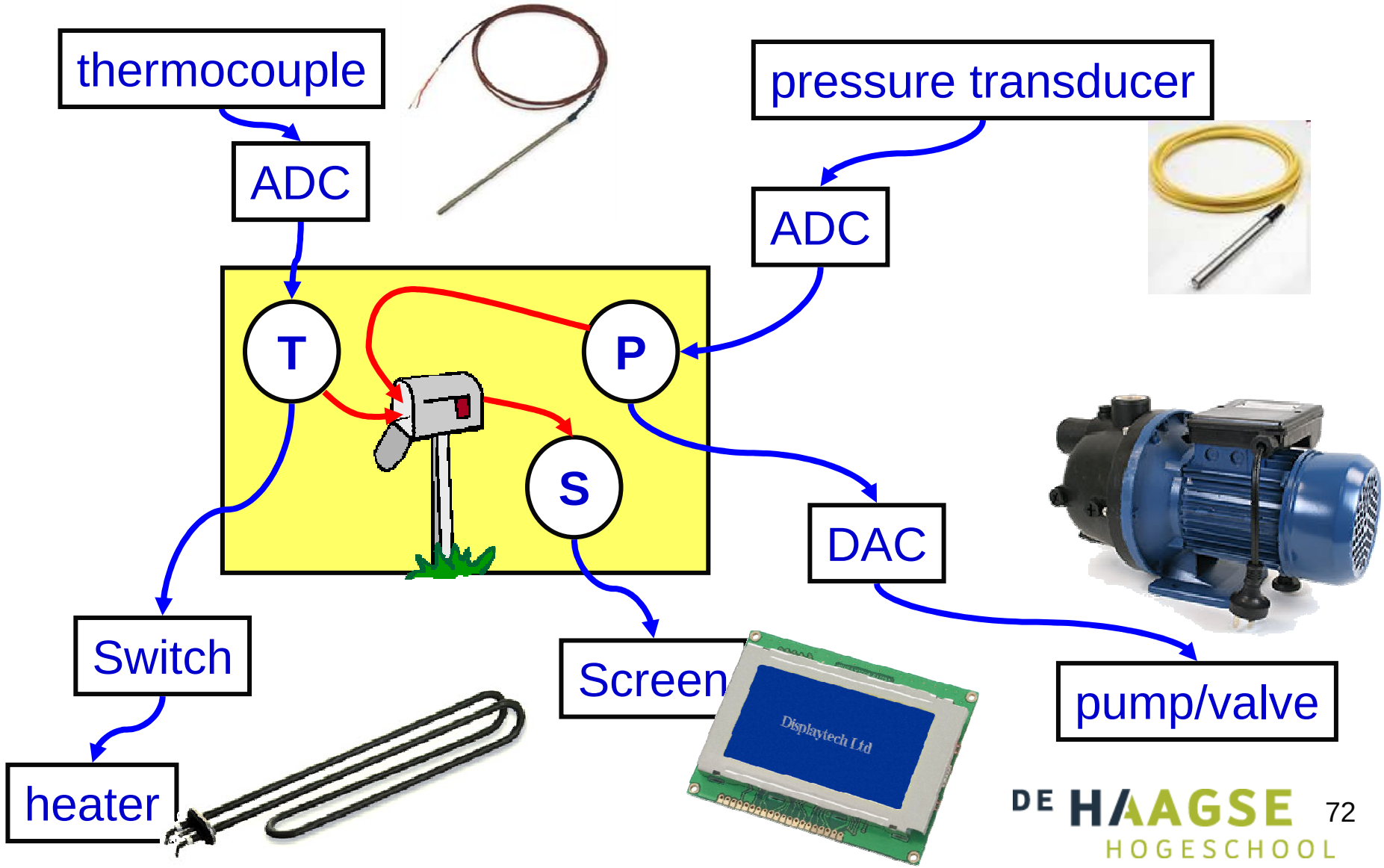
Synchronized Threads

```
Temp::Temp(boost::mutex& m): temp(0), switch_(false), display(m) { }
void Temp::operator()() {
    while (1) {
        read();
        {
            boost::lock_guard<boost::mutex> lock(display);
            std::cout<<"temperature = “<<std::fixed<<std::setw(4)
                        <<std::setprecision(1)<<temp<<", ";
            control();
            std::cout<<"switch = "<<switch_<<std::endl;
        }
        write();
        boost::this_thread::sleep(boost::posix_time::seconds(3));
    }
}
```

Synchronized Threads

```
int main() {
    boost::mutex m;
    Temp t(m);
    Pres p(m);
    boost::thread t1(boost::ref(t));
    boost::thread t2(boost::ref(p));
    t1.join();
    t2.join();
    std::cin.get();
    return 0;
}
```

Voorbeeld embedded system



Message Queue

```
#include <mqueue.h>
// ...
void* tempThread(void* p) {
    double temp; int switch_; mqd_t m; char buffer[128]; int i;
    check_errno( m=mq_open("/mq_par74", O_WRONLY) );
    for (i=0; i<10; i++) {
        int n=0;
        temp=readTemp();
        n=snprintf(buffer, sizeof buffer, "temperature = %4.1lf, ", temp);
        switch_=tempControl(temp);
        snprintf(&buffer[n], sizeof buffer-n, "switch = %d\n", switch_);
        check_errno( mq_send(m, buffer, sizeof buffer, 2) );
        writeSwitch(switch_);
        sleep(3);
    }
    check_errno( mq_close(m) );
    return NULL;
}
```

Zie volgende sheet…

Message Queue

```
void* presThread(void* p) {
    double pres, dac; mqd_t m; char buffer[128]; int i;
    check_errno( m=mq_open("/mq_par74", O_WRONLY) );
    for (i=0; i<10; i++) {
        int n=0;
        pres=readPres();
        n=snprintf(buffer, sizeof buffer, “pressure = %4.1lf, ", pres);
        dac=presControl(pres);
        snprintf(&buffer[n], sizeof buffer-n, "DAC = %5.1lf\n", dac);
        check_errno( mq_send(m, buffer, sizeof buffer, 3) );
        writeDAC(dac);
        sleep(1);
    }
    check_errno( mq_close(m) );
    return NULL;
}
```

Zie volgende sheet…

Message Queue

```
void* dispThread(void* p) {
    mqd_t m; struct mq_attr ma; char buffer[128]; int doorgaan=1;
    check_errno( m=mq_open("/mq_par74", O_RDONLY) );
    while (doorgaan) {
        do {
            check_errno( mq_receive(m, buffer, sizeof buffer, NULL) );
            if (strcmp(buffer, "CLOSE")==0)
                doorgaan=0;
            else
                printf(buffer);
                check_errno( mq_getattr(m, &ma) );
        }
        while (ma.mq_curmsgs>0);
        sleep(5);
        putchar('\007');
    }
    check_errno( mq_close(m) );
}
```

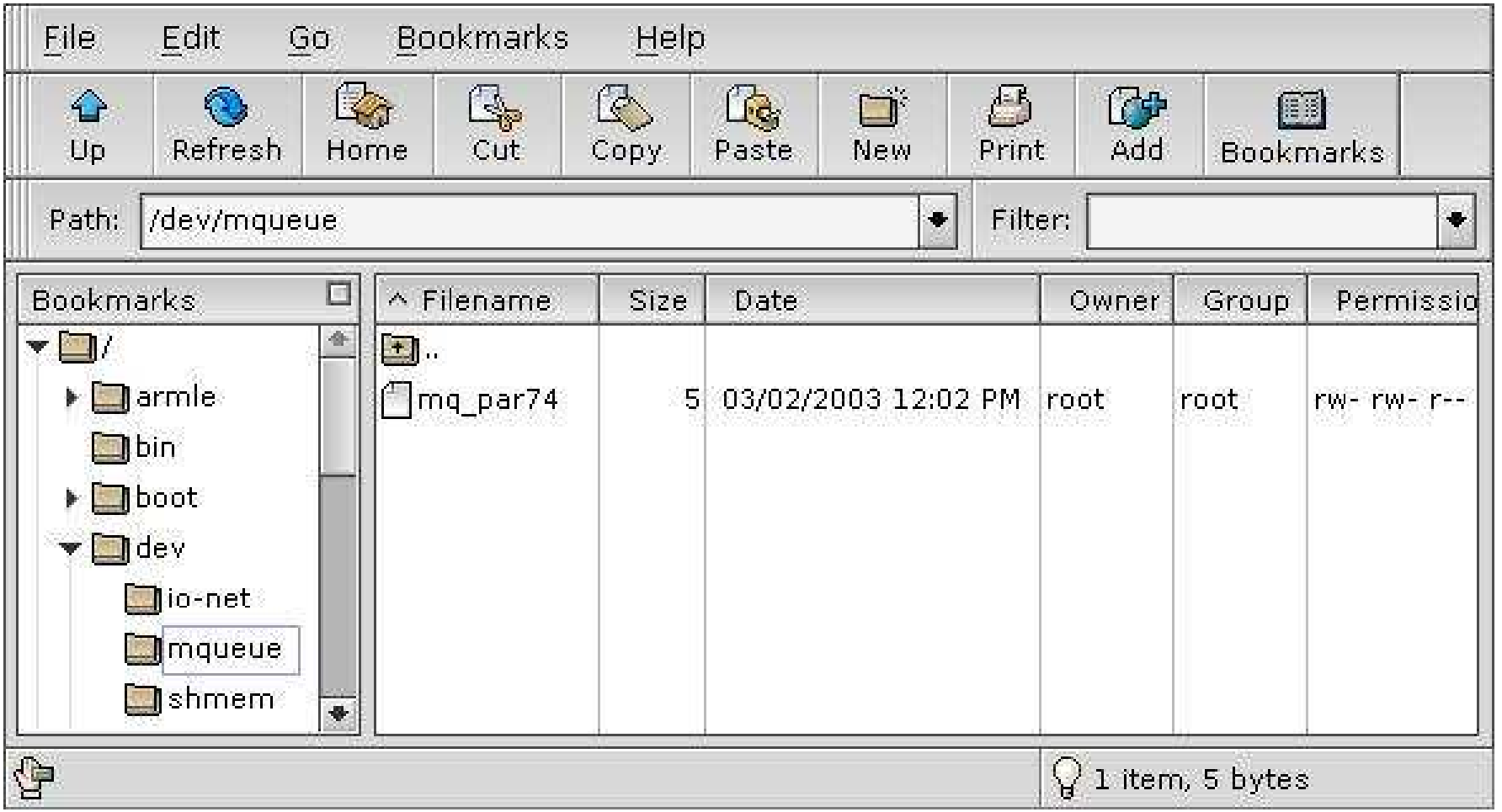
Zie volgende sheet…

Message Queue

```
int main(void) {
    pthread_t t1, t2, t3; mqd_t m; struct mq_attr ma;
    ma.mq_maxmsg=40;
    ma.mq_msgsize=128;
    check_errno(
        m=mq_open("/mq_par74", O_CREAT|O_EXCL|O_RDWR, 0666, &ma)
    );
    check( pthread_create(&t1, NULL, tempThread, NULL) );
    check( pthread_create(&t2, NULL, presThread, NULL) );
    check( pthread_create(&t3, NULL, dispThread, NULL) );
    check( pthread_join(t1, NULL) );
    check( pthread_join(t2, NULL) );
    check_errno( mq_send(m, "CLOSE", 6, 1) );
    check( pthread_join(t3, NULL) );
    check_errno( mq_close(m) );
    check_errno( mq_unlink("/mq_par74") );
    return EXIT_SUCCESS;
}
```

```
BEEP!
pressure = 15.6, DAC =  -5.6
pressure = 15.2, DAC =  -5.2
pressure = 14.8, DAC =  -4.8
pressure = 14.4, DAC =  -4.4
temperature =  2.5, switch = 0
```

/dev/mqueue


File
Edit
Go
Bookmarks
Help
Up
Refresh
Home
Cut
Copy
Paste
New
Print
Add
Bookmarks
Path:
/dev/mqueue
Filter:
Bookmarks
/
armle
bin
boot
dev
io-net
mqueue
shmem
Filename
Size
Date
Owner
Group
Permissio
..
mq_par74
5
03/02/2003 12:02 PM
root
root
rw- rw- r--
1 item, 5 bytes