IPC

inter process communication and synchronization

- shared variabele based (H8)
- message based (H9)
  - kan ook gebruikt worden in systemen zonder gedeeld geheugen (gedistribueerde systemen zie ESWE1C2).
  - POSIX: message queue
  - QNX: Neutrino kernel is volledig message based. Zie H2 QNX boek.
  - JAVA: Geen expliciete ondersteuning voor messages. J2EE heeft wel JMS Java Messages Service (ondersteund MOM zie ESWE1C2).

THRijswijk

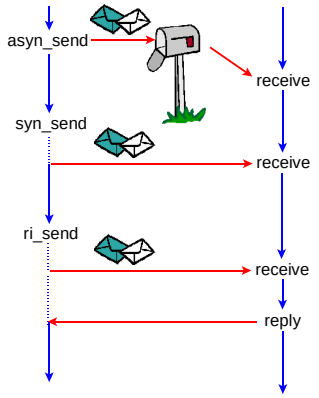
Messages

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.

THRijswijk

Messages



THRijswijk

Messages

synchroon met behulp van 2 asynchrone messages

Proces 1	Proces 2
asyn_send(mes)	receive(mes)
receive(ack)	async_send(ack)

remote invocation met behulp van 4 asynchrone messages

Proces 1	Proces 2
asyn_send(mes)	receive(mes)
receive(ack)	async_send(ack)
receive(reply)	//... construct reply
async_send(ack)	async_send(reply)
	receive(ack)

remote invocation met behulp van 2 synchrone messages

Proces 1	Proces 2
syn_send(mes)	receive(mes)
receive(reply)	//... construct reply
	sync_send(reply)

THRijswijk

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.

THRijswijk

Messages

Adressering

- direct: sender geeft receiver proces (of thread) op.
- indirect: sender geeft port, channel of mailbox op. (Kan ook synchron.)
- symetrisch: receiver geeft sender, port, channel of mailbox op.
- asymetrisch: receiver geeft niets op. (Client-server benadering.)

THRijswijk

Messages

Inhoud

- Tussen threads:
  - geen beperkingen
- Tussen processen:
  - geen pointers (elk proces heeft zijn eigen memory map).
- Tussen machines:
  - mogelijk problemen met representatie:
    - character codering
    - big-endian, little-endian

THRijswijk

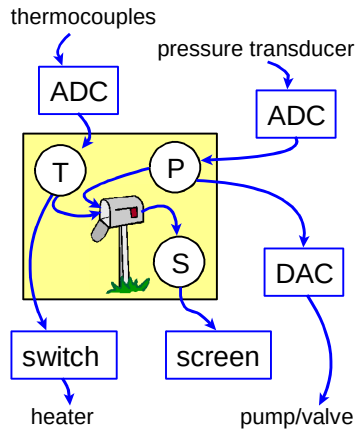
POSIX

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

THRijswijk

Voorbeeld



THRijswijk

mqueue

```
#include <mqueue.h>
// ...

void* tempThread(void* p) {
   double temp;
   int switch_;
   mqd_t m;
   char buffer[128];
   int i;
   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_=tempConvert(temp);
      snprintf(&buffer[n], sizeof buffer-n,
         "switch = %d\n", switch_);
      mq_send(m, buffer, sizeof buffer, 2);
      writeSwitch(switch_);
      sleep(3);
   }
   mq_close(m);
   return NULL;
}
```

check's van return waarden verwijderd om functie op sheet te passen!

Zie volgende sheet

THRijswijk

mqueue

```
void* presThread(void* p) {
   double pres, dac;
   mqd_t m;
   char buffer[128];
   int i;
   m=mq_open("/mq_par74", O_WRONLY);
   for (i=0; i<30; i++) {
      int n=0;
      pres=readPres();
      n=snprintf(buffer, sizeof buffer,
         "pressure = %4.1lf", pres) );
      dac=presConvert(pres);
      snprintf(&buffer[n], sizeof buffer-n, ",
         DAC = %5.1lf\n", dac);
      mq_send(m, buffer, sizeof buffer, 3);
      writeDAC(dac);
      sleep(1);
   }
   mq_close(m);
   return NULL;
}
```

Zie volgende sheet

THRijswijk

mqueue

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

Zie volgende sheet

THRijswijk

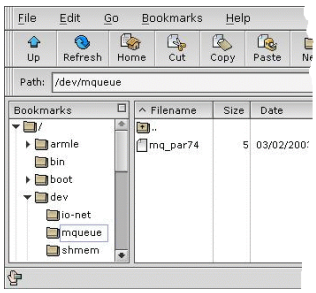
mqueue

```
int main(void) {
   pthread_t p1, p2, p3;
   mqd_t m;
   struct mq_attr ma;
   ma.mq_maxmsg=40;
   ma.mq_msgsize=128;
   m=mq_open("/mq_par74",
      O_CREAT|O_EXCL|O_RDWR, 0666, &ma);
   pthread_create(&p1, NULL, tempThread, NULL);
   pthread_create(&p2, NULL, presThread, NULL);
   pthread_create(&p3, NULL, dispThread, NULL);
   pthread_join(p1, NULL);
   pthread_join(p2, NULL);
   mq_send(m, "CLOSE", 6, 1);
   pthread_join(p3, NULL);
   mq_close(m);
   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
```

THRijswijk

/dev/mqueue



THRijswijk