3.8 Kommunikation über Nachrichten

- Klassen
  - Kanäle (pipes)
  - Briefkästen (queues)
  - Kommunikationsendpunkte (sockets)
- Übertragungsrichtung
  - unidirektional
  - bidirektional (voll-duplex, halb-duplex)
- Adressierung
  - implizit: UNIX-Pipes
  - explizit: CSP, OCCAM
  - globale Adressierung: Sockets, Ports
  - Gruppenadressierung: multicast, broadcast
  - funktionale Adressierung: Dienste
- Protokollfamilien
  - stromorientiert
  - nachrichtenorientiert
- Klassifikation
  - blockierend - nichtblockierend
  - gepuffert - ungepuffert
  - zuverlässig - unzuverlässig

UNIX-queues

- Initialisierung

```
#include "shared_file.h"
#include <sys/msg.h>
int main()
{ shared_int msqid(3008);
  key_t key = 1;

  msqid = msgget(key, int(0600|IPC_PRIVATE|IPC_CREAT));

  msgctl(msqid, IPC_RMID, 0);

  sleep(10);

  msqid = msgget(key, int(0600|IPC_PRIVATE|IPC_CREAT));

  return 1;
}
```

- Prozeß 0 (Erzeuger)

```
PROCESS(0)
  shared_int msqid(3008); char item[200]; int length;
  struct msgform
    { long mtype; char mtext[200];} buffer;
  while (1) {
    length = msgrcv(msqid,(msgbuf *)&buffer,200,1,0);
    cout<<"Producer received: "<<buffer.mtext
        <<"  sends: "<<flush;
    cin >> item;
    strcpy(buffer.mtext, item); buffer.mtype = 2;
    msgsnd(msqid,(msgbuf *)&buffer,strlen(item)+1,0);
    }
  }
PROCESS_END
```

- Prozeß 1 (Verbraucher)

```
PROCESS(1)
  shared_int msqid(3008); char item[200]; int i,
length;
  struct msgform
    { long mtype; char mtext[200];} buffer;
  buffer.mtype = 1; strcpy(buffer.mtext, "empty");
  for (i = 0; i < 3; i++)
    msgsnd(msqid,(msgbuf *)&buffer,
           strlen(buffer.mtext)+1,0);

  while (1) {
    length = msgrcv(msqid,(msgbuf *)&buffer,200,2,0);
    cout<<"Consumer received: "<<buffer.mtext
        <<"\n"<<flush;
    strcpy(buffer.mtext, "empty"); buffer.mtype = 1;
    msgsnd(msqid,(msgbuf *)&buffer,
           strlen(buffer.mtext)+1,0);
    sleep(1);
  }
PROCESS_END
```

UNIX-sockets

- Initialisierung

```
#include "shared_file.h"
int main()
{ unlink("consumer_socket");
  unlink("producer_socket");
  return 1;
}
```

- Prozeß 0 (Erzeuger)

```
PROCESS(0)
  char item[200];
  int sd;
  struct sockaddr_un producer_socket, consumer_socket;
  producer_socket.sun_family = AF_UNIX;
  strcpy(producer_socket.sun_path, "producer_socket");
  consumer_socket.sun_family = AF_UNIX;
  strcpy(consumer_socket.sun_path, "consumer_socket");
  unlink("producer_socket");

  sd = socket(AF_UNIX, SOCK_DGRAM, 0);
  bind(sd, (struct sockaddr *) &producer_socket,
           strlen(producer_socket.sun_path)
             + sizeof(producer_socket.sun_family));
  while (connect(sd,
                 (struct sockaddr *)&consumer_socket,
                 strlen(consumer_socket.sun_path)
                   + sizeof(consumer_socket.sun_family))
         < 0) {
    cerr << "\nError in connecting
         << "producer to consumer: "
         << sys_errlist[errno] << "\n"
         << "Trying again\n" << flush;
    sleep(1);
  }

  while (1) {
    recv(sd, item, 199, 0);
    cout << "Producer received: " << item
         << "\n" << flush;
    cout << "Message to be sent by the producer: "
         << flush;
    cin >> item;
    send(sd, item, strlen(item) + 1, 0);
  }
PROCESS_END
```

- Prozeß 1 (Verbraucher)

```
PROCESS(1)
  char item[200];
  int sd, i;
  struct sockaddr_un producer_socket, consumer_socket;
  producer_socket.sun_family = AF_UNIX;
  strcpy(producer_socket.sun_path, "producer_socket");
  consumer_socket.sun_family = AF_UNIX;
  strcpy(consumer_socket.sun_path, "consumer_socket");
  unlink("consumer_socket");

  sd = socket(AF_UNIX, SOCK_DGRAM, 0);
  bind(sd, (struct sockaddr *) &consumer_socket,
           strlen(consumer_socket.sun_path)
             + sizeof(consumer_socket.sun_family));
  while (connect(sd,
                  (struct sockaddr *) &producer_socket,
                  strlen(producer_socket.sun_path)
                    + sizeof(producer_socket.sun_family))
         < 0) {
    cerr << "\nError in connecting
         << "consumer to producer: "
         << sys_errlist[errno] << "\n"
         << "Trying again\n" << flush;
    sleep(1);
  }
  for (i = 0; i < 3; i++) {
    strcpy(item, "Please send next item");
    send(sd, item, strlen(item) + 1, 0);
  }
  while(1) {
    recv(sd, item, 199, 0);
    cout << "Consumer received: "
         << item << "\n" << flush;
    strcpy(item, "Please send next item");
    send(sd, item, strlen(item) + 1, 0);
  }
PROCESS_END
```