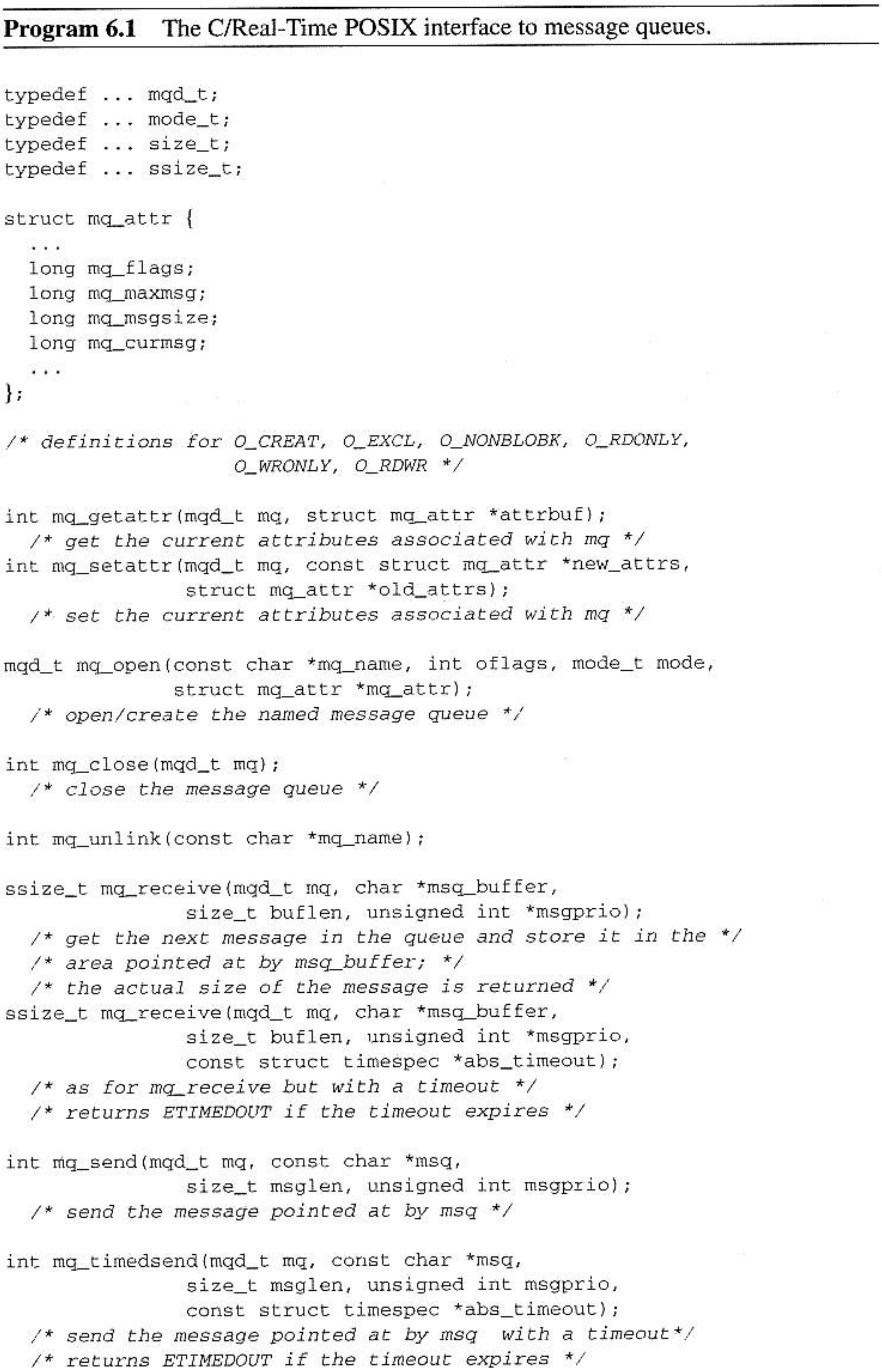
Beispiele zu Kapitel 4: Kommunikation und Synchronisation – Nachrichten (*Messages*)

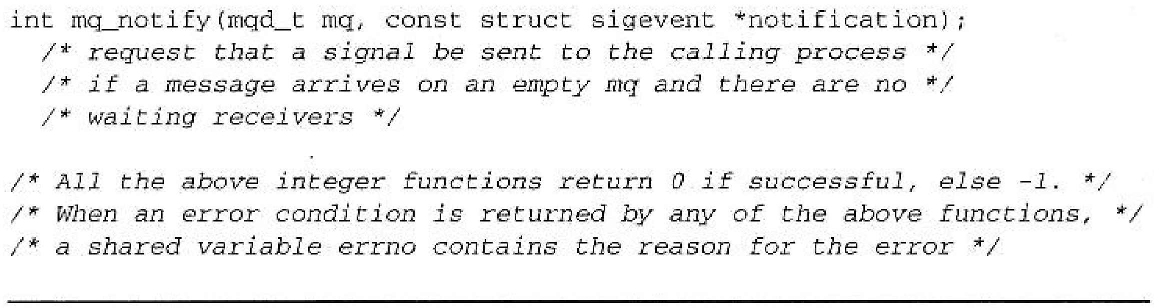
Aus: Alan Burns, Andy Wellings: *Real-Time Systems and Programming Languages. Ada, Real-Time Java and C/Real-Time POSIX*. Addison Wesley, 2009. (Kapitel 6)

Beispiel 4-1: C/Real-Time POSIX-Schnittstelle für Nachrichten-Warteschlangen (*message queues*)



Program 6.1 The C/Real-Time POSIX interface to message queues.

```
typedef ... mqd_t;
typedef ... mode_t;
typedef ... size_t;
typedef ... ssize_t;

struct mq_attr {
  ...
  long mq_flags;
  long mq_maxmsg;
  long mq_msgsize;
  long mq_curmsg;
  ...
};

/* definitions for O_CREAT, O_EXCL, O_NONBLOBK, O_RDONLY,
                   O_WRONLY, O_RDWR */

int mq_getattr(mqd_t mq, struct mq_attr *attrbuf);
  /* get the current attributes associated with mq */
int mq_setattr(mqd_t mq, const struct mq_attr *new_attrs,
               struct mq_attr *old_attrs);
  /* set the current attributes associated with mq */

mqd_t mq_open(const char *mq_name, int oflags, mode_t mode,
              struct mq_attr *mq_attr);
  /* open/create the named message queue */

int mq_close(mqd_t mq);
  /* close the message queue */

int mq_unlink(const char *mq_name);

ssize_t mq_receive(mqd_t mq, char *msq_buffer,
               size_t buflen, unsigned int *msgprio);
  /* get the next message in the queue and store it in the */
  /* area pointed at by msq_buffer; */
  /* the actual size of the message is returned */
ssize_t mq_receive(mqd_t mq, char *msq_buffer,
               size_t buflen, unsigned int *msgprio,
               const struct timespec *abs_timeout);
  /* as for mq_receive but with a timeout */
  /* returns ETIMEDOUT if the timeout expires */

int mq_send(mqd_t mq, const char *msq,
               size_t msglen, unsigned int msgprio);
  /* send the message pointed at by msq */

int mq_timedsend(mqd_t mq, const char *msq,
               size_t msglen, unsigned int msgprio,
               const struct timespec *abs_timeout);
  /* send the message pointed at by msq  with a timeout*/
  /* returns ETIMEDOUT if the timeout expires */
```



```
int mq_notify(mqd_t mq, const struct sigevent *notification);
  /* request that a signal be sent to the calling process */
  /* if a message arrives on an empty mq and there are no */
  /* waiting receivers */

/* All the above integer functions return 0 if successful, else -1. */
/* When an error condition is returned by any of the above functions, */
/* a shared variable errno contains the reason for the error */
```

Beispiel 4-2: Programmierung des einfachen Roboterarms (vgl. Kapitel 3)

```
typedef enum {xplane, yplane, zplane} dimension;

void move_arm(dimension D, int P);

#define DEFAULT_NBYTES 4
/* assume that the coordinate can be represented as 4 characters */
int nbytes = DEFAULT_NBYTES;


#define MQ_XPLANE  "/mq_xplane" /* message queue name */
#define MQ_YPLANE  "/mq_yplane" /* message queue name */
#define MQ_ZPLANE  "/mq_zplane" /* message queue name */
#define MODE ...                /* mode information for mq_open */
/* names of message queues */

void controller(dimension dim) {
  int position, setting;
  mqd_t my_queue;
  struct mq_attr ma;
  char buf[DEFAULT_NBYTES];
  ssize_t len;

  position = 0;
  switch(dim) { /* open appropriate message queue */
    case xplane:
      my_queue = MQ_OPEN(MQ_XPLANE, O_RDONLY, MODE, &ma);
      break;
    case yplane:
      my_queue = MQ_OPEN(MQ_YPLANE, O_RDONLY, MODE, &ma);
      break;
    case zplane:
      my_queue = MQ_OPEN(MQ_ZPLANE, O_RDONLY, MODE, &ma);
      break;
    default:
      return;
    };

  while (1) {
    /* read message */
    len = MQ_RECEIVE(my_queue, &buf[0], nbytes, NULL);
    setting = *((int *) (&buf[0]));
    position = position + setting;
    move_arm(dim, position);
  };
}
```

```
int main(int argc, char **argv) {
  mqd_t mq_xplane, mq_yplane, mq_zplane;
    /* one queue for each process */
  struct mq_attr ma; /* queue attributes */
  int xpid, ypid, zpid;
  char buf[DEFAULT_NBYTES];

    /* set the required message queue attributes */
    ma.mq_flags = 0;    /* No special behaviour */
    ma.mq_maxmsg = 1;
    ma.mq_msgsize = nbytes;

    /* calls to set the actual attributes for the */
    /* three message queues */

    mq_xplane = MQ_OPEN(MQ_XPLANE, O_CREAT|O_EXCL, MODE, &ma);
    mq_yplane = MQ_OPEN(MQ_YPLANE, O_CREAT|O_EXCL, MODE, &ma);
    mq_zplane = MQ_OPEN(MQ_ZPLANE, O_CREAT|O_EXCL, MODE, &ma);

    /* Duplicate the process to get the three controllers */
    switch (xpid = FORK()) {
        case 0:     /* child */
                controller(xplane);
                exit(0);
        default:    /* parent */
            switch (ypid = FORK()) {
              case 0:     /* child */
                controller(yplane);
                exit(0);
              default:    /* parent */
                switch (zpid = FORK()) {
                  case 0:     /* child */
                    controller(zplane);
                    exit(0);
                  default:    /* parent */
                    break;
          }
      }
  }


  while (1) {
    /* find new position and set up buffer to transmit each
       coordinate to the controllers, for example */
    MQ_SEND(mq_xplane, &buf[0], nbytes, 0);
  }
}
```