

### PR: Programmiersprache C (Linux) Einheit 9: IPC, Make

Roland A. Eggetberger

Inst. f. Informationsverarbeitung und Mikroprozessortechnik (FIM)  
Johannes Kepler Univ. Linz, Altenbergerstr. 69, A-4040 Linz, Austria

### IPC

Pipes  
Signals  
Message Queues

#### Pipes (I)

- Formatiert (I)

```
#include <stdio.h>
FILE *popen(const char *command,
            const char *type);
int pclose(FILE *stream);
```

```
command ... Prozess
type ..... Lesen ("r") oder
          Schreiben ("w")
```

#### Pipes (II)

- Formatiert (II)

Schreiben auf den Stream ist ein Schreiben auf `stdin` des Prozesses  
Der Prozess verwendet dann dieselbe `stdout`  
Analog erfolgt das Lesen  
Dabei wird formatierte I/O verwendet

#### Pipes (III)

- Low-Level (I)

```
#include <unistd.h>
int pipe(int filedes[2]);
```

Legt Filedeskriptoren an

```
fd[0] ... Lesen
fd[1] ... Schreiben
```

#### Pipes (IV)

- Low-Level (II)

Lesen, Schreiben und Schliessen der Filedeskriptoren erfolgt mit Low-Level-Operationen (`read, write, close`)

Zur Kommunikation zwischen Parent- und Child-Prozess

### Signals (I)

- Senden

```
#include <sys/types.h>
#include <signal.h>
int kill(pid_t pid, int sig);
```

Sende Signal **sig** an Prozess mit PID **pid**

### Signals (II)

- Signale (I)

SIGHUP	1 Hangup
SIGINT	2 Interrupt
SIGQUIT	3 Quit
SIGILL	4 Illegal Instruction
SIGABRT	6 Abort signal from abort
SIGFPE	8 Floating point exception
SIGKILL	9 Kill signal
SIGSEGV	11 Invalid memory reference

### Signals (III)

- Signale (II)

SIGUSR1	10 User-defined signal 1
SIGUSR2	12 User-defined signal 2
SIGPIPE	13 Broken pipe
SIGALRM	14 Timer signal from alarm
SIGTERM	15 Termination signal
SIGCONT	18 Continue if stopped
SIGSTOP	19 Stop process
SIGCHLD	20 Child stopped or terminated

### Signals (IV)

- Signalbehandlung (I)

Default-Aktion ausführen lassen  
 Signal blocken (falls möglich)  
 Signal abfangen

### Signals (V)

- Signalbehandlung (II)

```
#include <signal.h>
void (*signal(int signum,
              void (*handler)(int)))(int);
```

Für den Handler gibt es drei Möglichkeiten

SIG_DFL ... Defaultverhalten
SIG_IGN ... Ignorieren
Eine eigene Funktion

### Message Queues (I)

- Idee

Möglichkeit zum erweiterten Informationsaus-  
 tausch zwischen Prozessen

Jeder Message Queue ist eine eigener Typ  
 zugeordnet

Der Zugriff auf die Message Queue erfolgt über  
 einen Key

### Message Queues (II)

- Initialisierung (I)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget(key_t key, int msgflg)
```

Rückgabewert ist eine Message Queue ID, bzw.  
im Fehlerfall -1

Mit `msgflg` werden Rechte gesetzt

### Message Queues (III)

- Initialisierung (II)

Mögliche Werte für `key`

`IPC_PRIVATE` ... Privat für den Prozess

Mögliche Werte für `msgflg`

`IPC_CREAT` ..... Neu anlegen

`IPC_EXCL` ..... Anlegen, falls nicht  
existierend

### Message Queues (IV)

- Initialisierung (III)

```
#include <sys/types.h>
#include <sys/ipc.h>
key_t ftok(char *pathname, int proj)
```

Rückgabe eines Keys aus einem Pfad und einer  
Integerzahl

### Message Queues (V)

- Handhabung (I)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgctl(int msqid, int cmd,
           struct msqid_ds *buf)
```

In Abhängigkeit von `cmd` werden verschiedene  
Aktionen durchgeführt

### Message Queues (VI)

- Handhabung (II)

Mögliche Kommandos

<code>IPC_STAT</code> ... Status der Queue abfragen
<code>IPC_SET</code> .... Rechte ändern
<code>IPC_RMID</code> ... Queue entfernen

### Message Queues (VII)

- Statusstruktur (I)

```
struct msqid_ds {
    struct ipc_perm msg_perm;
    /* operation permission */
    struct msg *_msg_first;
    /* pointer first message on queue */
    struct msg *_msg_last;
    /* pointer last message on queue */
```

### Message Queues (VIII)

- Statusstruktur (II)

```
__time_t msg_stime;
    /* time of last msgsnd command */
__time_t msg_rtime;
    /* time of last msgrecv command */
__time_t msg_ctime;
    /* time of last change */
struct wait_queue *__wwait;
struct wait_queue *__rwait;
```

### Message Queues (IX)

- Statusstruktur (III)

```
unsigned short int __msg_cbytes;
    /* current number of bytes on queue */
unsigned short int msg_qnum;
    /* number of messages on queue */
unsigned short int msg_qbytes;
    /* number of bytes allowed on queue */
__ipc_pid_t msg_lspid; /*last msgsnd()*/
__ipc_pid_t msg_lrpid; /*`n`msgrecv()*/};
```

### Message Queues (X)

- Senden (I)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgsnd(int msqid, struct msgbuf *msgp,
           int msgsz, int msgflg)
```

### Message Queues (XI)

- Senden (II)

Dabei wird ein Message-Puffer verwendet

```
struct msgbuf {
    long mtype;
    /* message type, must be >0 */
    char mtext[MSGSZ]; /* message data */
};
```

### Message Queues (XII)

- Senden (III)

Das Argument `msgflg` steuert das Senden

```
IPC_NOWAIT ... Falls die Queue ihr
Speicherlimit erreicht hat
wird nicht gesendet
Sonst ..... wird gewartet
```

### Message Queues (XIII)

- Empfangen (I)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgrcv(int msqid, struct msgbuf *msgp,
           int msgsz, long msgtyp,
           int msgflg)
```

### Message Queues (XIV)

- Empfangen (II)

Beim Empfangen muss zusätzlich ein `msgtyp` angegeben werden

```
= 0 ... Erste Message wird gelesen
> 0 ... Lese erste Message vom msgtyp
< 0 ... Lese erste Message von <= -msgtyp
```

### Message Queues (XV)

- Empfangen (III)

Bedeutung von `msgflg` beim Empfangen

```
MSG_EXCEPT .... Invertieren des Lesens bei
msgtyp > 0
MSG_NOERROR ... Abschneiden der Message
falls Text länger als
msgsz
```

### Make

Make  
Makefile  
Kommentar  
Makros  
Spezielle Regeln  
Optionen

### Make (I)

- Kommandogenerator
- Organisation einer Softwareentwicklung

### Make (II)

- Kommando

```
make [-f file][options][targets][macros]
file ... Name des Makefiles
```

### Makefile (I)

- Kommentar
- Makros
- Spezielle Regeln
- Defaults

### Makefile (II)

- Beispiel

```
hello: hello.c
        gcc -g -o hello hello.c
```

Dabei werden die Files nur im Bedarfsfall neu erzeugt

### Kommentar

- Verwendung

```
# Kommentar
```

### Makros (I)

- Definition

```
name = data
```

Verwendung

```
$(name)
```

### Makros (II)

- Vordefinierte Makros

```
CC ..... Compilerkommando
CFLAGS ... Compileroptionen
OBJ ..... Objektfiles
```

Dabei ist zu beachten, dass die Compileroptionen vorangestellt werden

### Makros (III)

- Kommandozeile

```
make myprog DIR=/home/myhome
```

Aufeinanderfolgende Definitionen unter Anführungszeichen

### Makros (IV)

- Stringersetzung

```
MYSRCS = dieses.c jenes.c
MYOBJS = $(MYSRCS:.c=.o)
```

Erfolgt nur am Ende eines Makros und vor Whitespaces

### Makros (V)

- Interne Makros

```
$@ ... Aktuelles Zielfile
$? ... Files neuer als Zielfile
```

### Spezielle Regeln (I)

- Bedingte Ausführung

```
target: source [source2] [source3] ...
      command
      [command2]
      [command3]
      ...
```

```
target ..... Name des Zielfiles
commands ... Ein Tab eingerückt
```

### Spezielle Regeln (II)

- Unbedingte Ausführung

```
source:
      command
      [command2]
```

### Spezielle Regeln (III)

- Kommandofolgen

Jedes Kommando wird quasi in einer eigenen Shell ausgeführt  
Abhilfe

```
command1; \
command2
```

### Spezielle Regeln (IV)

- Endungsregel

```
.c.o:
$(CC) $(CFLAGS) -c $<
```

Falls keine andere Regel definiert wurde,  
kommen Endungsregeln zum Einsatz

```
$< ... Ersetzungsfile
$* ... Ersetzungsfile ohne Endung
```

### Spezielle Regeln (V)

- Dummies

```
updated: file1 file2
update $?
touch $@
```

Es wird nur der Zeitstempel aktualisiert

### Optionen (I)

- Anzeigeoptionen

```
-p ... Anzeigen aller Makros und Regeln  
-d ... Debugging Informationen  
-n ... Kommandos nur anzeigen  
-s ... Kommandos nur ausführen
```

### Optionen (II)

- Sonstige

```
-f ... Angabe eines Filenamen  
-o ... Direktes Einbinden älterer Files  
-r ... Implizite Regeln vernachlässigen  
-t ... Nur Zeitstempel updaten
```