1.1 Systemprogrammierung unter UNIX

UNIX ist ein *Multi-User-/Multi-Tasking*-System, das als Mehrzweck-Betriebssystem für vielerlei Aufgaben eingesetzt wird und für seine Flexibilität, Modularität, leichte Wartbarkeit und vor allem höchste Stabilität bekannt ist.

Eine Domäne von UNIX ist traditionell – neben Kommunikation und Textverarbeitung – die Softwareentwicklung. Schon im Standardlieferumfang eines UNIX-Systems sind zahlreiche Werkzeuge zur Entwicklung und Pflege von Programmen in den unterschiedlichsten Programmiersprachen (C, C++, Pascal, FORTRAN, COBOL, ADA, Smalltalk etc.) enthalten. Entwicklungsumgebungen mit integrierter Source-Code-Verwaltung, graphischem Debugger und diversen Analysatoren wie z.B. SGI's *Developer Magic* lassen beim Entwickler von System- und Applikationssoftware für das UNIX-System selbst, aber auch für andere Zielsysteme keine Wünsche mehr offen.

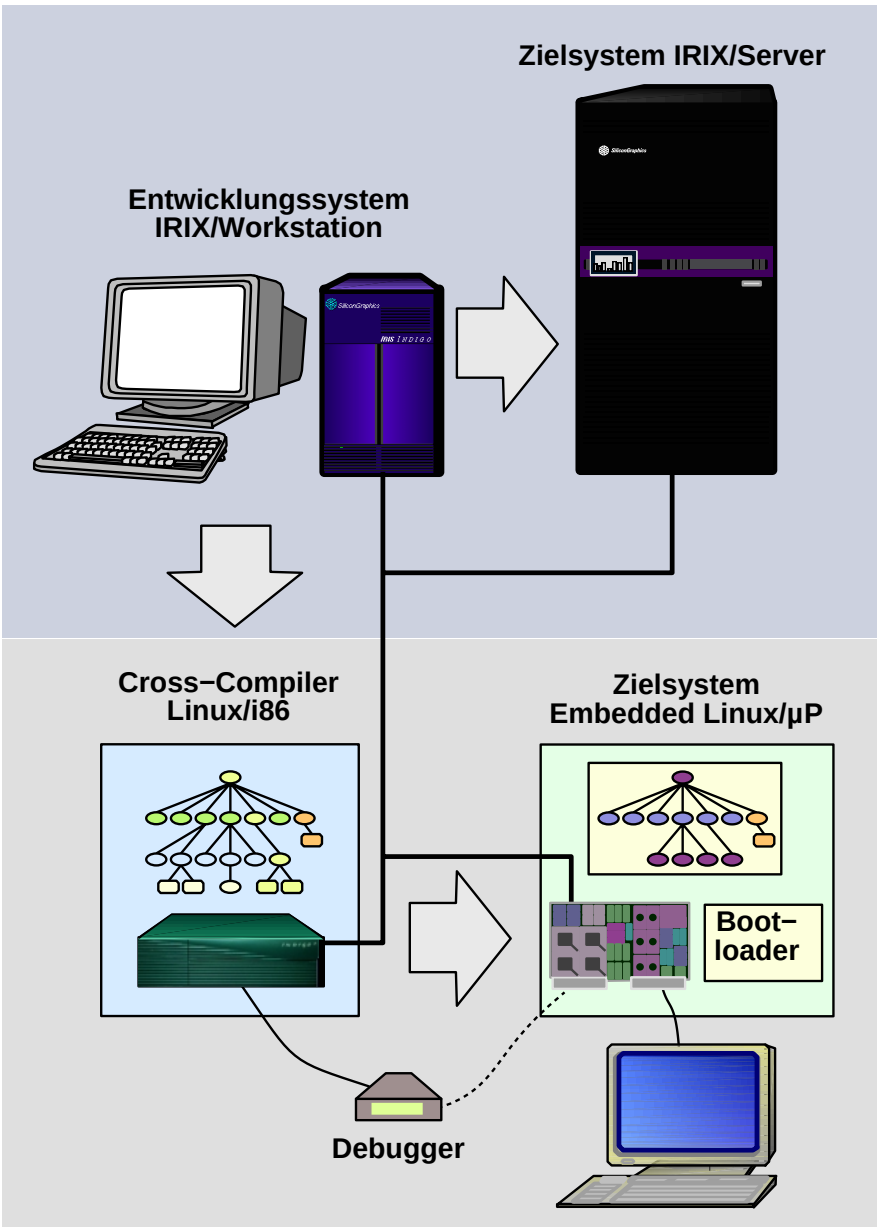
UNIX wird auch für viele andere Aufgaben eingesetzt, wo es auf Leistung bei günstigem Preis ankommt: GNU/Linux, das einem modernen PC die Leistungsfähigkeit einer ansonsten sehr teuren Workstation verleiht, erfreut sich zunehmender Beliebtheit – auch bei Herstellern von Servern (SGI, Sun) bis hin zu den Mainframes (IBM), aber auch im Bereich der Embedded Systems. NetBSD- und FreeBSD-Systeme werden wegen ihrer hohen Performanz und Sicherheit oft als Internet-Server eingesetzt.

Die Grenzen zwischen den diversen UNIX-Systemen verwischen dabei immer mehr: So können auf einer Workstation von SGI mit derem (proprietären) Betriebssystem IRIX z.B. der KDE-Desktop verwendet und andere Linux-Applikationen installiert werden oder umgekehrt auf Linux-Systemen diverse IRIX-Tools vorhanden sein wie `gr_osview`, `top`, *OpenGL*, *GLUT*, *XFS-Dateisystem*, *Samba* usw.

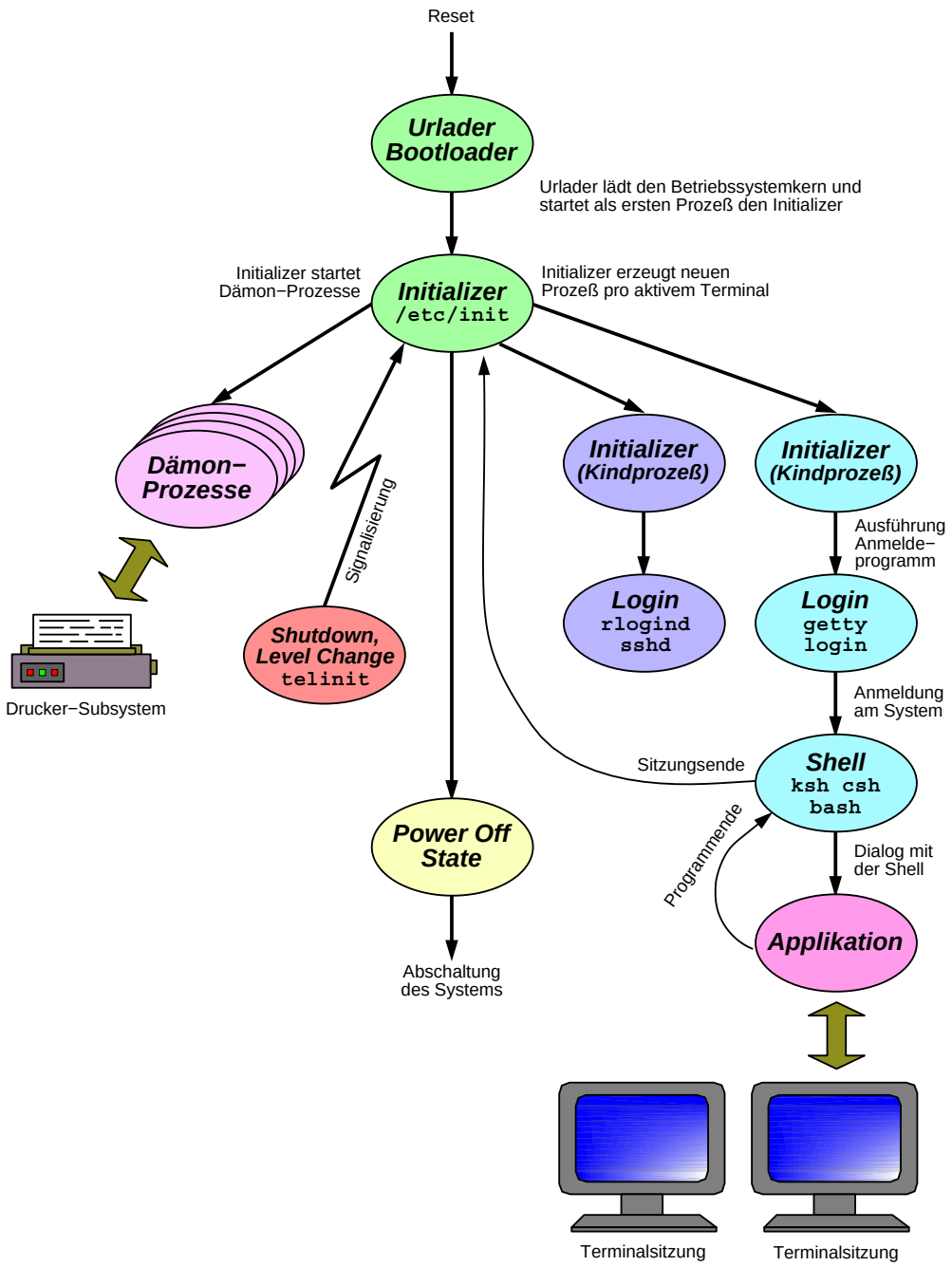
Als Programmiersprache für die Beispiele und Musterlösungen in diesem Kurs wird »ANSI-C« eingesetzt, obwohl die Funktionen des Betriebssystemkerns auch in jeder anderen Sprache verwendet werden können, sofern entsprechende Schnittstellenfunktionen vorhanden sind. Die Referenzplattformen, auf denen die Beispielprogramme dieses Kurses getestet wurden, sind eine SGI O2 mit IRIX 6.5, eine SGI 1400L mit RedHat Linux, ein PC/i86 mit SuSE Linux 8.2 sowie ein Starter-Kit STK8/PowerPC mit Embedded Linux aus dem ELDK 3.1.1.

Zwischen UNIX und der Programmiersprache »C« besteht eine sehr enge Verwandschaft, denn »C« wurde ursprünglich dafür entwickelt, um UNIX auf verschiedene Rechnersysteme portieren zu können. Neben der »C«-Standardbibliothek, in der häufig benötigte Funktionen wie z.B. für Datei-Ein-/Ausgabe etc. auf einer höheren Abstraktionsebene gemäß dem ANSI-Standard zusammengefaßt sind, steht Programmen, die auf einem UNIX- (oder jedem anderem POSIX-kompatiblen) System ablaufen, der gesamte Leistungsumfang des Betriebssystemkerns über **Systemaufrufe** zur Verfügung.

Systemprogrammierung unter UNIX [3]

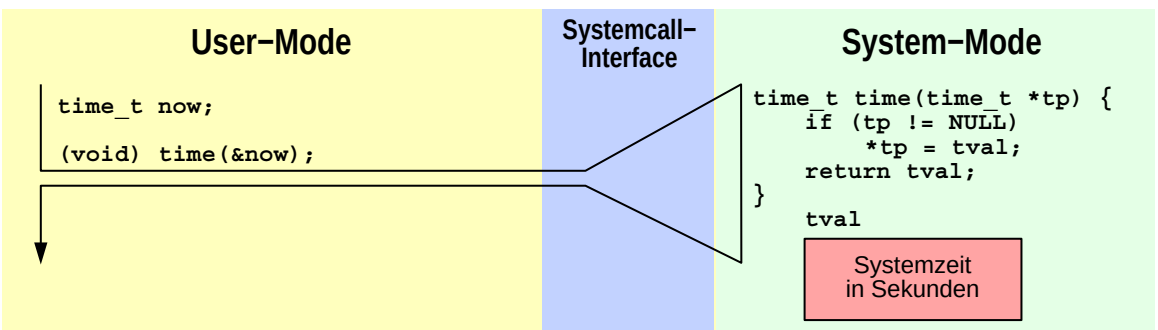


Realisierung des Multi-User-Betriebsmodus [11]

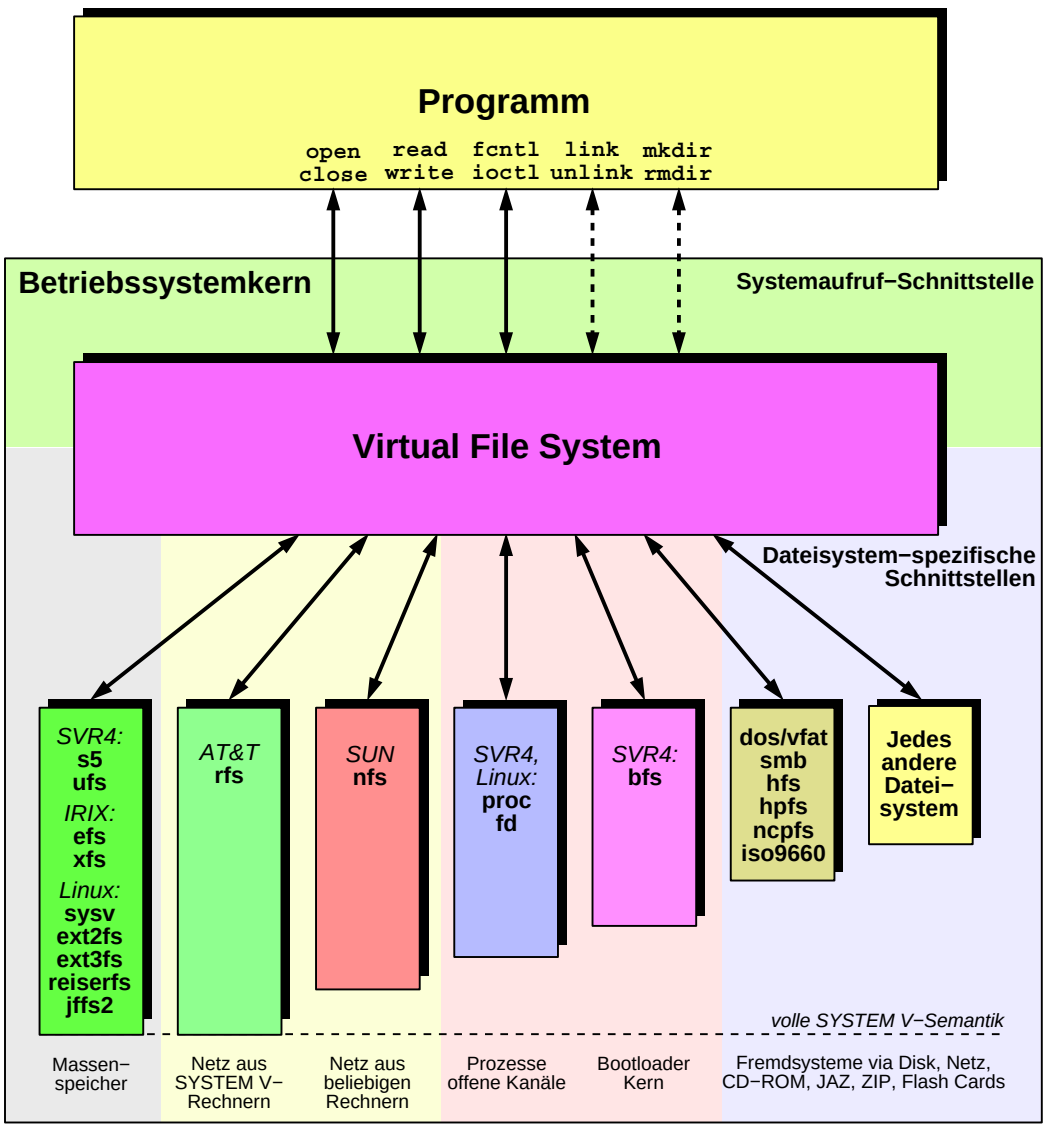


Ausführung von Systemaufrufen [19]

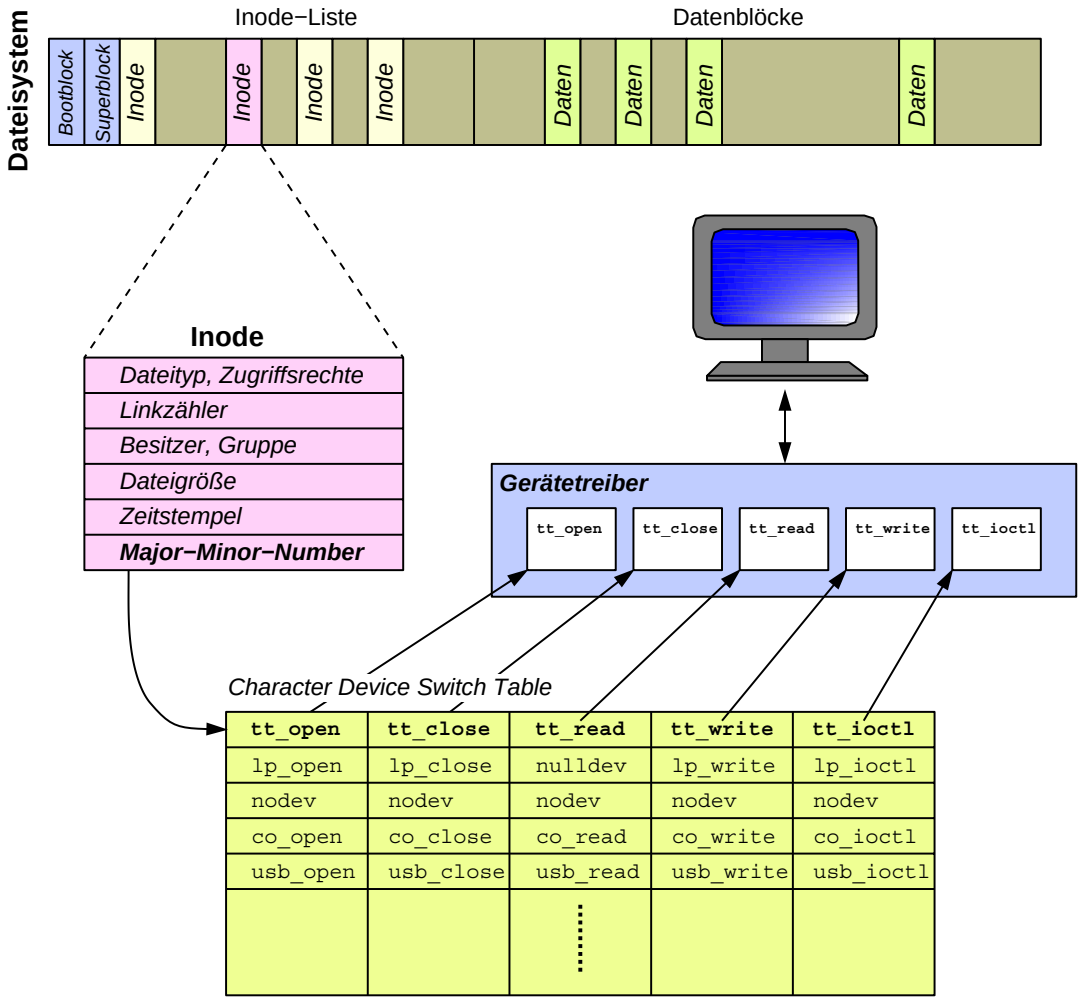
```
 1= #include <stdio.h>                    /* fuer printf(3) */
 2= #include <stdlib.h>                   /* fuer exit(2) */
 3= #include <sys/types.h>                /* fuer time_t */
 4= #include <time.h>                     /* fuer ctime(3) und time(2) */
 5=
 6= int main(void) {
 7=     time_t now;                       /* Sekunden seit 1. Jan 1970 */
 8=
 9=     /*
10=     ** Kalenderzeit ermitteln, in ASCII-String
11=     ** konvertieren und ausgeben.
12=     */
13=     if (time(&now) != (time_t)-1)
14=         (void) printf("Today is: %s", ctime(&now));
15=
16=     return 0;
17= }
```



Das Virtual File System (VFS) [41]



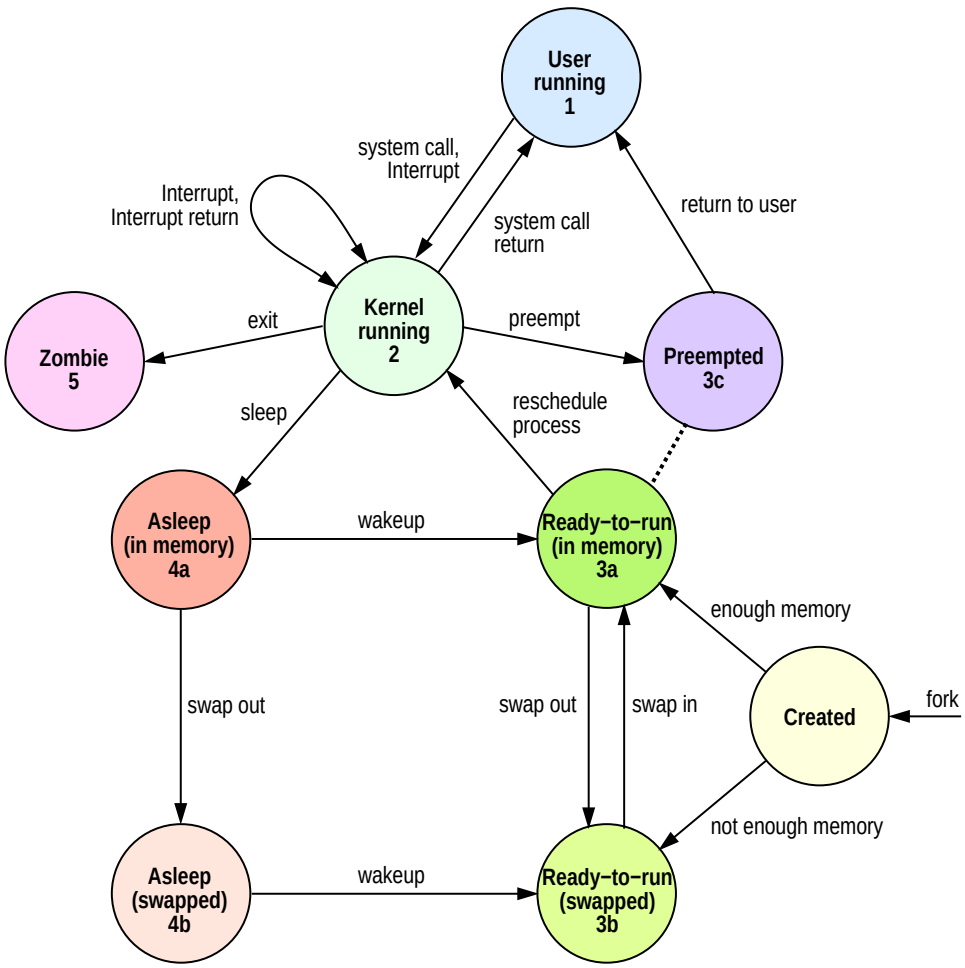
Implementation von Spezialdateien [49]



Daten lesen mit read [55]

```
 1= #include <stdio.h>                      /* fuer printf(3)/perror(3) */
 2= #include <stdlib.h>                     /* fuer exit(2) */
 3= #include <string.h>                     /* fuer strcpy(3) */
 4= #include <unistd.h>                     /* fuer exit(2)/close(2) */
 5= #include <sys/types.h>                  /* fuer abstrakte Datentypen */
 6= #include <sys/stat.h>                   /* fuer open(2) */
 7= #include <fcntl.h>                      /* fuer open(2) */
 8= #include <errno.h>                      /* fuer strerror(3) */
 9=
10= int main(int ac, char **av) {           /* Datei oeffnen und ausgeben */
11=     char lbuf[1024];                    /* Zwischenspeicher */
12=     ssize_t nread;                      /* Anzahl gelesener Zeichen */
13=     int fd, cnt = 0;                    /* File-Deskriptor, Zaehler */
14=
15=     if (ac == 1) {                      /* wenn keine Argumente angegeben, */
16=         fd = STDIN_FILENO;              /* ... dann 'stdin' verarbeiten */
17=         (void) printf("reading STDIN\n");
18=     } else if (ac == 2) {               /* ... andernfalls Datei eroeffnen */
19=         fd = open(*++av, O_RDONLY);
20=         if (fd < 0)                     /* Fehlerbehandlung */
21=             syserr("open failed");
22=         (void) printf("open returned %d\n", fd);
23=     } else {                            /* Fehlbedienung */
24=         (void) fprintf(stderr, "Usage: %s [file]\n", av[0]);
25=         exit(2);
26=     }
27=     do {                                /* Daten lesen */
28=         nread = read(fd, lbuf, sizeof(lbuf)-1);
29=         if (nread == 0)                 /* EOF erkannt */
30=             (void) strcpy(lbuf, "EOF DETECTED");
31=         else if (nread < 0) {           /* Fehler bei read() */
32=             (void) strcpy(lbuf, "READ FAILED");
33=             (void) strcat(lbuf, strerror(errno));
34=         } else                          /* Zeile terminieren */
35=             lbuf[(size_t)nread] = '\0'; /* fuer printf() */
36=         (void) printf("read %d returned %d: ", ++cnt, (int)nread);
37=         pr_buf(lbuf);                   /* Zeichenkette anzeigen */
38=     } while (nread > 0);                /* solange Daten gelesen */
39=
40=     /* Nach EOF erneut lesen, um Rueckkehrwert von read() zu demonstrieren */
41=     nread = read(fd, lbuf, sizeof(lbuf)-1);
42=     (void) printf("read beyond EOF returned %d\n", (int)nread);
43=
44=     if ((cnt=close(fd)) < 0)            /* Datei schliessen */
45=                 syserr("close failed");
46=     (void) printf("close returned %d\n", cnt);
47=     return 0;
48= }
```

Zustände eines Prozesses [113]



Asynchron Meldungen ausgeben [141]

```
19= int main(void) {
20=     char lbuf[1024], *ep;              /* Speicher fuer Eingabe */
21=     unsigned int delay;                /* Zeitverzoegerung in Sekunden */
22=     ssize_t nread;                     /* Anzahl gelesener Zeichen */
23=     pid_t pid;                         /* PID des Kindprozesses */
24=
25=     (void) setbuf(stdout, NULL);       /* Pufferung ausschalten */
26=     (void) sigset(SIGCHLD, chexits);   /* Zombies abfangen */
27=
28= #define EVER    ;;
29=     for (EVER) {                       /* Eingabe lesen, Kind starten */
30=         (void) printf("Enter delay and message: ");
31=         if ((nread=read(0, lbuf, sizeof(lbuf)-1)) <= 0) {
32=             if (nread < 0) {
33=                 if (errno == EINTR)    /* Ignoriere Unterbrechungen */
34=                     continue;          /* durch sich beendende Kinder */
35=                 perror("read failed");
36=             }
37=             break;                     /* Abbruch bei EOF/Fehler */
38=         }
39=         if (lbuf[nread-1] == '\n')     /* abschliessendes NL entfernen */
40=             nread--;
41=         lbuf[nread] = '\0';            /* Zeichenkette terminieren */
42=
43=         delay = (unsigned int)strtoul(lbuf, &ep, 10);
44=         if (ep > lbuf && (*ep == ' ' || *ep == '\t') && *++ep != '\0') {
45=             (void) sighold(SIGCHLD);   /* Signale blockieren */
46=
47=             if ((pid=fork()) < 0)      /* Kindprozess erzeugen */
48=                 syserr("fork failed");
49=
50=             if (!pid) {                /* SIGCHLD ignorieren, Prozess */
51=                 (void) sigset(SIGCHLD, SIG_DFL); /* SIGCHLD ignorieren */
52=                 sleep(delay);          /* Prozess suspendieren */
53=                 (void) printf("\n[%u sec] %s\a\n", delay, ep);
54=                 exit(0);               /* ... und terminieren */
55=             }
56=             (void) sigrelse(SIGCHLD);  /* Erzeuger entsperrt Signale */
57=         } else
58=             (void) fprintf(stderr, "Invalid input, try again\n");
59=     }
60=     (void) putchar('\n');              /* abschliessendes Newline */
61=     return 0;
62= }
```

Anzeige von Priorität und nice-Wert mit ps(1) [155]

IRIX auf SGI O2:

```
$ ps -el
  F S   UID    PID   PPID  C PRI NI  P    SZ:RSS     WCHAN TTY    TIME CMD
  0 S     0      1      0  0  20 20  *   425:131  805310a0 ?      0:45 init
  4 S     0     85      1  0  20 20  *   419:117  805900a0 ?      0:04 syslogd
  4 S     0    551      1  0  88 RT  *  5547:436  805cd0a0 ?      0:00 soundsche
  4 S     0    565      1  0  30 10  *  1323:454  805e50a0 ?      0:06 xwsh
  0 S   100    574    565  0  20 20  *   146:95   803f1d6c pts/0  0:00 sh
  4 S   100    637      1  0  20 20  *  2467:983  805e36a0 ?     71:52 apanel
  0 S     0    641    551  0  88 RT  *  5547:435  c0762900 ?      0:00 soundsche
  0 R   100  53845    630  0  20 20  0   429:138         - pts/4  0:00 ps
...
```

SuSE Linux 8.2 auf i686 PC:

```
$ PS_PERSONALITY=sgi ps -el
F S   UID   PID  PPID  C PRI  NI P    SZ:RSS  WCHAN  TTY      TIME CMD
0 S     0     1     0  0  75   0 *   125:244  schedu ?        0:05 init
0 S     0     4     1  0  94  19 *     0:0    ksofti ?        0:00 ksoftirqd
0 S     0   154     1  0  60 -20 *     0:0    down_i ?        0:00 lvm-mpd
0 S     0   750     1  0  75   0 *   364:600  schedu ?        0:00 syslogd
0 S     0   753     1  0  75   0 *   577:1356 syslog ?        0:00 klogd
0 R   500  2505  2431  0  81   0 0   903:1576 -      pts/3    0:00 ps
...
```

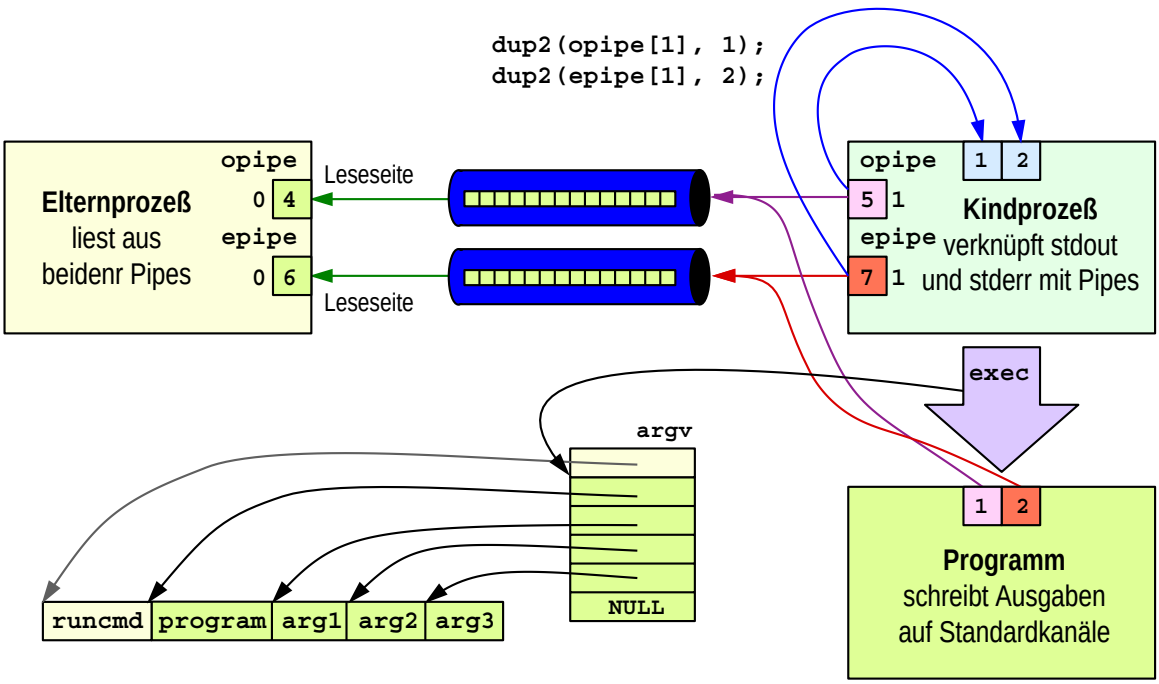
Embedded Linux 2.4.25 auf TQM860L PowerPC:

```
$ PS_PERSONALITY=sgi ps -el
  F S UID    PID  PPID  C PRI  NI P   SZ:RSS  WCHAN  TTY    TIME CMD
100 S root     1     0  0  69   0 *  371:548  do_sel ?      0:02 init [3]
040 S root     2     1  0  69   0 *    0:0    contex ?      0:00 [keventd]
040 S root     3     1  0  79  19 *    0:0    r_next ?      0:00 [ksoftirqd_CPU0]
040 S root     4     1  0  69   0 *    0:0    swap_o ?      0:00 [kswapd]
040 S root     5     1  0  69   0 *    0:0    kupdat ?      0:00 [bdflush]
040 S root     6     1  0  69   0 *    0:0    kupdat ?      0:00 [kupdated]
040 S root     7     1  0  69   0 *    0:0    neigh_ ?      0:00 [mtdblockd]
040 S root     8     1  0  69   0 *    0:0    unswap ?      0:00 [rpciod]
140 S root   229     1  0  69   0 *  460:736  do_sel ?      0:00 syslogd -m 0
140 S root   233     1  0  69   0 *  370:496  do_sys ?      0:00 klogd -x
140 S bin    280     1  0  69   0 *  443:688  poll   ?      0:00 portmap
140 S root   303     1  0  69   0 *  735:920  do_sel ?      0:00 xinetd -stayalive
100 S root   309     1  0  69   0 *  882:1184 wait4  ?      0:00 login -- root
100 S root   310   309  0  69   0 *  655:1408 wait4  ttyS0  0:00 -bash
100 R root   384   310  0  77   0 0  774:1020 -      ttyS0  0:01 ps -efl
$
```

Signalreaktion einrichten mit signal [165]

```
 1= #include <stdio.h>                     /* fuer printf(3)/perror(3) */
 2= #include <unistd.h>                    /* fuer sleep(3) */
 3= #include <stdlib.h>                    /* fuer exit(2) */
 4= #include <signal.h>                    /* fuer signal(2) */
 5=
 6= static int volatile maxloop = 15;      /* Schleifendurchgaenge in main */
 7=
 8= static void onint(int sig) {           /* Signalbehandlungsfunktion */
 9=     (void) signal(sig, onint);         /* Signal-Reaktion re-installieren */
10=
11=     clearerr(stdout);                  /* Prophylaktisch Flag loeschen */
12=     if (sig == SIGQUIT) {              /* Bei QUIT-Signal terminieren */
13=         (void) printf("\n-> got SIGQUIT, bye!\n"); /* Kontrollmeldung */
14=         exit(0);
15=     }
16=     (void) printf("\n-> got SIGINT\n"); /* Kontrollmeldung */
17=     maxloop = 15;                      /* Zaehler zuruecksetzen */
18=     return;                            /* Rueckkehr an unterbrochene Stelle */
19= }
20=
21= int main(void) {
22=     char *str;                         /* Hilfsvariable */
23=     void (*oldint)(int);               /* Vorherige Einstellung von SIGINT */
24=
25=     setbuf(stdout, NULL);              /* Zeilenpufferung ausschalten */
26=                                        /* Signalreaktion einstellen */
27=     if ((oldint=signal(SIGINT, onint)) == SIG_ERR ||
28=         signal(SIGQUIT, onint) == SIG_ERR) {
29=         perror("signal failed");
30=         exit(1);
31=     }
32=     if (oldint == SIG_DFL)             /* Meldung vorbereiten und ausgeben */
33=          str = "SIG_DFL";              /* Default-Reaktion */
34=     else if (oldint == SIG_IGN)
35=          str = "SIG_IGN";              /* Ignore-Reaktion */
36=     else str = "function()";           /* Benutzerdefinierte Reaktion */
37=
38=     (void) printf("old SIGINT = %s, new SIGINT = onint()\n", str);
39=     (void) printf("start main loop ");
40=
41=     while (maxloop-- > 0) {            /* Solange kein "Time-Out" */
42=         (void) sleep(2);               /* Verzoegerung und Kontrollmeldung */
43=         (void) printf("still running ... ");
44=     }
45=     (void) printf("\nmain loop timed out\n");
46=     return 0;
47= }
```

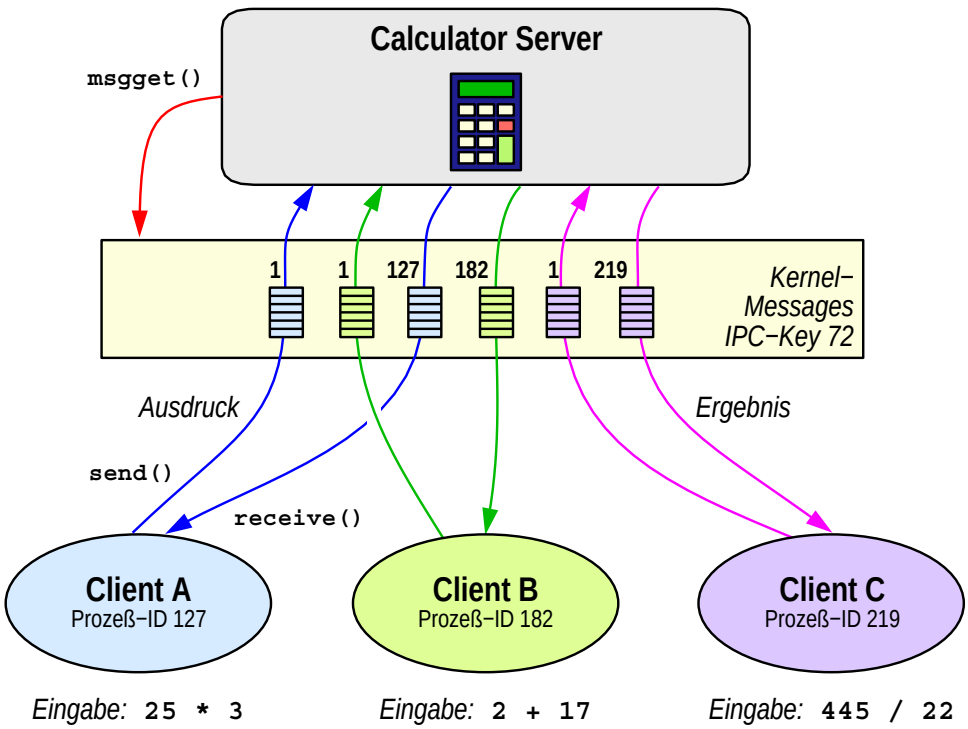
Mehrere Pipes zu einem Programm (I) [197]



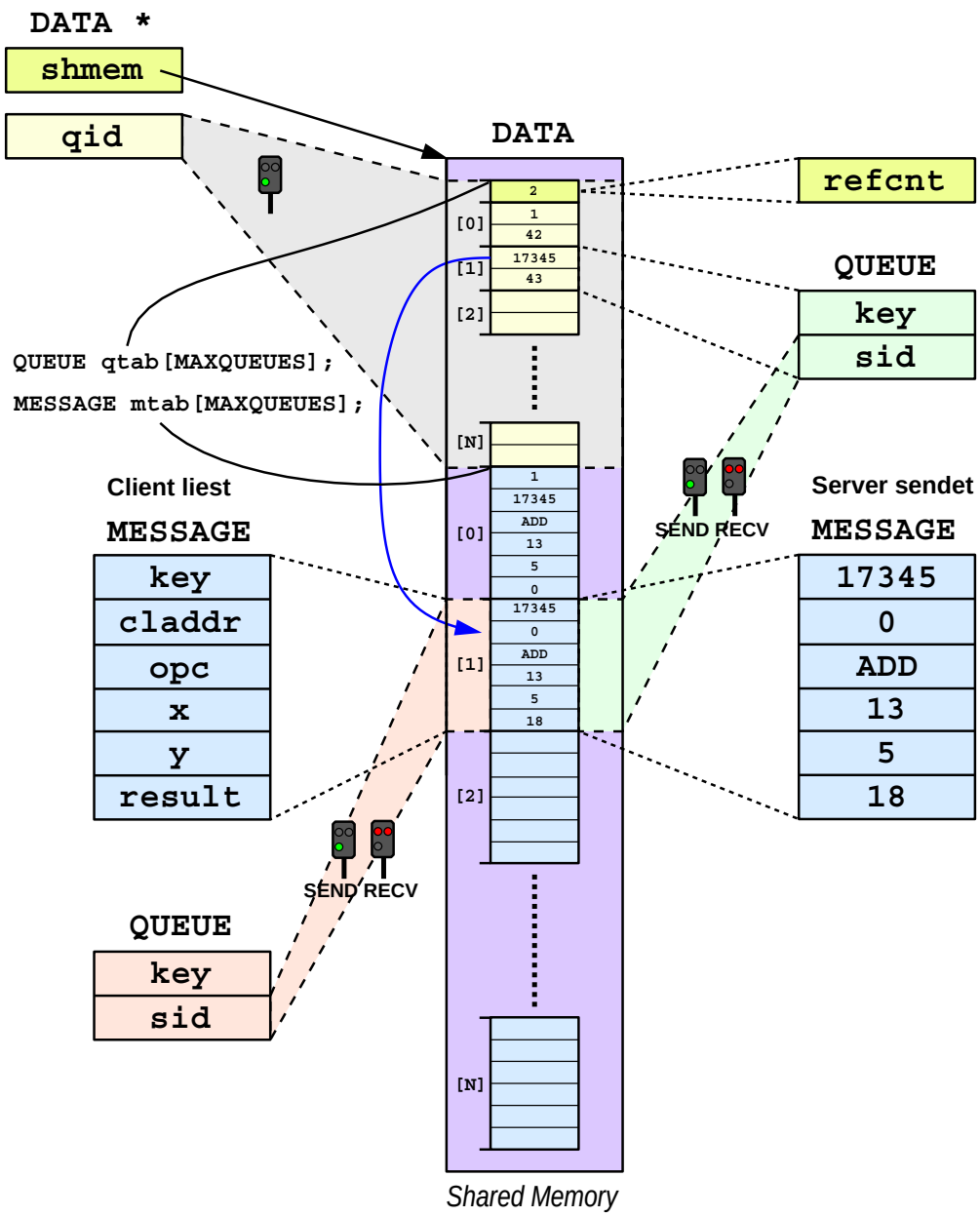
Mehrere Pipes zu einem Programm (II) [199]

```
13= int main(int ac, char **av) {
14=     pid_t pid;                          /* Prozess-ID Kindprozess */
15=     int opipe[2];                       /* FDs der Pipe fuer stdout */
16=     int epipe[2];                       /* FDs der Pipe fuer stderr */
17=     char lbuf[1024];                    /* Speicher fuer Eingaben */
18=     ssize_t nread = 1;                  /* Anzahl gelesener Zeichen */
19=     struct pollfd fds[2];               /* Struktur fuer zwei Kanaele */
20=
21=     if (ac < 2) {                       /* Argumente pruefen */
22=         (void) fprintf(stderr, "Usage: %s command [arg1 arg2 ... argN]\n", *av);
23=         exit(2);
24=     }                                   /* Zwei Pipes erstellen */
25=     if (pipe(opipe) < 0 || pipe(epipe) < 0)
26=         syserr("pipe failed");
27=     if ((pid=fork()) < 0)               /* Kindprozess erzeugen */
28=         syserr("fork failed");
29=     if (pid == 0) {                     /* Kind fuehrt Kommando aus */
30=         if (dup2(opipe[1], STDOUT_FILENO) < 0 || dup2(epipe[1], STDERR_FILENO) < 0)
31=             syserr("dup2 failed");
32=         /* Unbenoetigte Pipe-FDs schliessen */
33=         (void) close(opipe[0]); (void) close(opipe[1]);
34=         (void) close(epipe[0]); (void) close(epipe[1]);
35=         (void) execvp(av[1], &av[1]);   /* Programm ausfuehren */
36=         syserr("exec failed");
37=     }                                   /* Elternprozess liest Ausgaben */
38=     (void) close(epipe[1]);             /* unbenoetigte FDs schliessen */
39=     (void) close(opipe[1]);
40=     fds[0].fd = opipe[0];               /* stdout-Pipe vom Kind */
41=     fds[1].fd = epipe[0];               /* stderr-Pipe vom Kind */
42=     fds[0].events = fds[1].events = POLLIN|POLLHUP; /* Eingabe oder EOF */
43=
44=     while (nread > 0) {                 /* Solange nicht Dateiende */
45=         if (poll(fds, 2, -1) <= 0)      /* blockierend ueberwachen */
46=             syserr("poll failed");      /* Fehlerbehandlung */
47=
48=         if ((fds[0].revents&POLLIN) == POLLIN && /* Eingabe von stdout */
49=             (nread=read(fds[0].fd, lbuf, sizeof(lbuf)-1)) > 0) {
50=             lbuf[nread] = '\0';         /* Endekennung fuer printf */
51=             (void) printf("-- read from stdout:\n%s", lbuf);
52=         }
53=         if ((fds[1].revents&POLLIN) == POLLIN && /* Eingabe von stderr */
54=             (nread=read(fds[1].fd, lbuf, sizeof(lbuf)-1)) > 0) {
55=             lbuf[nread] = '\0';         /* Endekennung fuer printf */
56=             (void) printf("-- read from stderr:\n%s", lbuf);
57=         }
58=         if ((fds[0].revents&POLLHUP) == POLLHUP || /* EOF bei Pipe */
59=             (fds[1].revents&POLLHUP) == POLLHUP)
60=             break;
61=     }
62=     (void) close(epipe[0]);             /* FDs der Pipes schliessen */
63=     (void) close(opipe[0]);
64=     return 0;                           /* ... und terminieren */
65= }
```

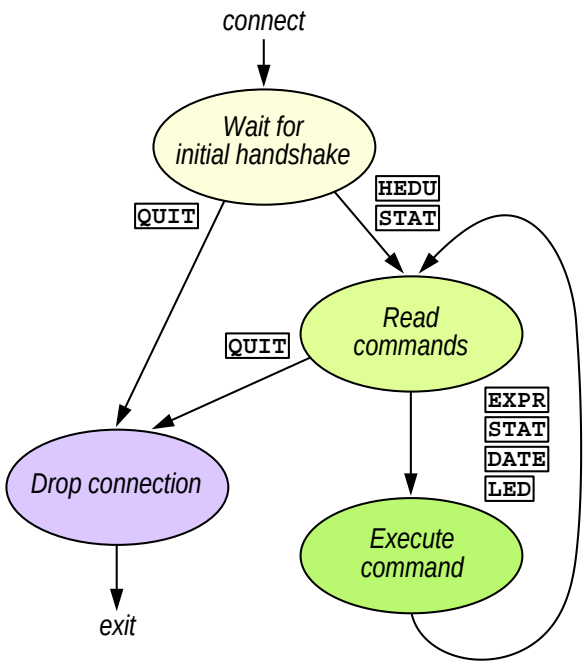
Architektur des »Calculator-Service« mit Kernel-Messages [245]



Architektur des »Calculator-Service« mit Shared-Memory [255]

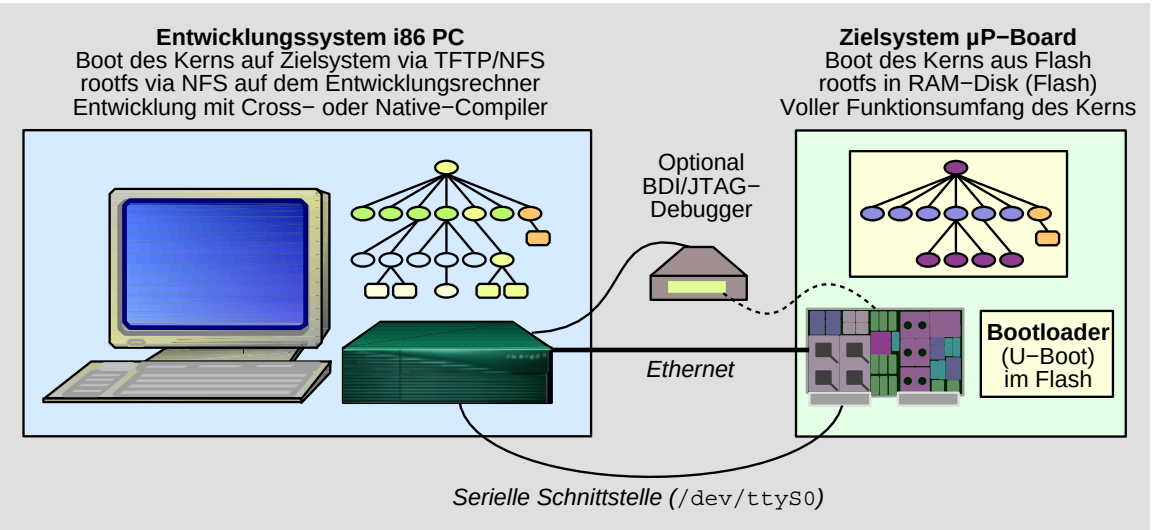


»Calculator-Service« mit BSD-Sockets [279]




```
$ telnet kaputnik.rent-a-guru.de 9000
Trying 194.123.185.135...
Connected to kaputnik.rent-a-guru.de.
Escape character is '^]'.
hedu
200 HIBOSS!
stat
216 STATUS connected 11 sec., idle 11 sec.
expr 237 * 168
210 RESULT 237 * 168 = 39816
expr 312 / 0
410 DIVZERO division by zero
quit
280 BYE!
Connection closed by foreign host.
$
```

Embedded Linux Developement Kit (ELDK) [319]




```
$ cu stk8
Connected.
U-Boot 1.1.3 (Apr  7 2005 - 17:21:52)
CPU:   XPC86xxxZPnnD4 at 50 MHz: 16 kB I-Cache 8 kB D-Cache FEC present
Board: TQM860L-AB.205
DRAM:  16 MB
FLASH:  8 MB
In:    serial
Out:   serial
Err:   serial
Net:   SCC ETHERNET, FEC ETHERNET
PCMCIA:   No Card found
Type "run flash_nfs" to mount root filesystem over NFS
Hit any key to stop autoboot:  0
=> sntp 192.168.1.75          Kalenderzeit einstellen
Date: 2005-04-14 Time: 10:56:29
=> run flash_nfs              Boot des Kerns aus Flash mit rootfs via NFS
oder
=> run flash_self             Boot des Kerns aus Flash mit rootfs aus RAM-Disk
```

INHALTSVERZEICHNIS