

```

Sep 19 14:20:18 amd64 sshd[20494]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61557
Sep 19 14:27:41 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 01:00:01 amd64 /usr/sbin/cron[29278]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 20 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 02:00:01 amd64 /usr/sbin/cron[30103]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 20 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:46:44 amd64 sshd[6516]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:48:41 amd64 sshd[609]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd[6094]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 20 13:27:25 amd64 sshd[9077]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64242
Sep 20 15:27:35 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:37:11 amd64 sshd[10102]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63375
Sep 20 16:37:11 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:38:10 amd64 sshd[10140]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63546
Sep 21 01:00:01 amd64 /usr/sbin/cron[17055]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 21 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 02:00:01 amd64 /usr/sbin/cron[19878]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 21 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:43:26 amd64 sshd[31088]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63397
Sep 21 17:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:53:39 amd64 sshd[31267]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64363
Sep 21 18:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 19:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 01:00:01 amd64 /usr/sbin/cron[19878]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 22 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[14599]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 22 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 20:23:21 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 01:00:01 amd64 /usr/sbin/cron[24739]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 23 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 02:00:01 amd64 /usr/sbin/cron[2355]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 23 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:05 amd64 sshd[6554]: Accepted publickey for esser from ::ffff:192.168.1.5 port 59771 ssh2
Sep 23 18:04:05 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:34 amd64 sshd[6066]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62093
Sep 24 01:00:01 amd64 /usr/sbin/cron[12436]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 24 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron[13253]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 24 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 11:15:48 amd64 sshd[20998]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
Sep 24 11:15:48 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 13:49:08 amd64 sshd[23197]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 13:49:08 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_midi_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 20:25:31 amd64 sshd[29399]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62566
Sep 24 20:25:31 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 01:00:02 amd64 /usr/sbin/cron[1862]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 25 01:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 02:00:01 amd64 /usr/sbin/cron[14841]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 25 02:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[8889]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:47 amd64 sshd[8921]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:30:02 amd64 sshd[9372]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62029
Sep 25 11:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:05:37 amd64 sshd[11554]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:05:37 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:06:10 amd64 sshd[11586]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd[11606]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:33 amd64 sshd[11630]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 15:25:33 amd64 sshd[12930]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62778

```

4. Prozesse

C-Funktionen zu Prozessen

- getpid(), getppid()
- Rückgabewert in wait (int *status)
- nice()
- setpgid(), setsid(), getpgid(), getsid()

Weitere Themen:

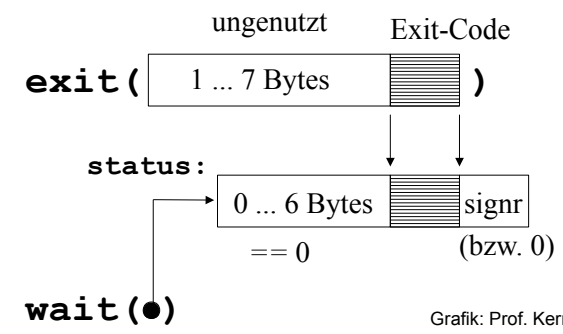
- /proc-Dateisystem
- Argumente (argc, argv), getopt()

Prozesse

Schon gesehen:

- Prozess-Hierarchie (fork, Vater/Sohn)
- Programm nachladen (exec)
- Warten auf Prozess (wait, waitpid)

Zusammenhang exit / wait



- auslesen des exit-Werts: über (status >> 8) oder mit Makro WEXITSTATUS
- Signalnummer != 0, falls erzwungener Abbruch durch Signal

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
```

```
void forktest (int retval) {
    int pid, status;
    pid = fork ();
    if (pid==0) {
        // child:
        if (retval >= 0) {
            // normal termination
            printf ("Child: exiting with exit code %d\n", retval);
            exit (retval);
        } else {
            // abnormal termination
            printf ("Child: exiting abnormally\n");
            abort ();
        }
    } else {
        // parent:
        printf ("Parent: Waiting for child to terminate\n");
        wait (&status);
        if (WIFEXITED(status)) {
            printf ("Parent: Child exited with exit code %d\n",
                WEXITSTATUS(status));
        } else {
            printf ("Parent: Child exited abnormally.\n");
        }
    }
}

int main () {
    forktest (0);
    forktest (3);
    forktest (-1); // terminate abnormally
    return 0;
}
```

```
esser@ubu64:forkwait$ gcc forkwait.c
esser@ubu64:forkwait$ ./a.out
Parent: Waiting for child to terminate
Child: exiting with exit code 0
Parent: Child exited with exit code 0
Parent: Waiting for child to terminate
Child: exiting with exit code 3
Parent: Child exited with exit code 3
Parent: Waiting for child to terminate
Child: exiting abnormally
Parent: Child exited abnormally.
```

nice(): Priorität ändern (2)

- Bei Verwendung:
 - #include <unistd.h>
- für folgenden Mini-Benchmark:
 - gettimeofday (#include <sys/timeb.h>)
 - verwendet Typ struct timeval:


```
struct timeval {
    time_t      tv_sec;      /* seconds */
    suseconds_t tv_usec;    /* microseconds */
};
```
 - sinf, cosf aus Mathe-Bibliothek (beim Kompilieren: -om angeben; #include <math.h>)

nice(): Priorität ändern (1)

- Der Aufruf nice(x) ändert die Priorität des Prozesses und gibt
 - die neue Priorität
 - oder -1 im Fehlerfall zurück
- gültige Werte für den Nice-Wert: -20 .. 19
 - Normalwert: 0
 - 1..19: niedrige Priorität („freundlich“)
 - -20..-1: hohe Priorität („unfreundlich“)
- Nur root darf höhere Prioritäten wählen

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/timeb.h>
#include <math.h>

void nicetest (int niceval) {
    struct timeval start, end;
    int i, j, k, pid, diff;
    float data[1000][1000]; float a, b;
    pid = fork ();
    if (pid==0) { // child
        nice (niceval);
        gettimeofday (&start, NULL);
        for (i=0; i<1000; i++) {
            for (j=0; j<1000; j++) { data[i][j] = (float)(i*j); }
        };
        for (k=0; k<10; k++) { // etwas Zeit verbrauchen...
            for (i=0; i<1000; i++) {
                for (j=0; j<1000; j++) {
                    a = data[i][j]; b = data[j][i]; data[i][j] = sinf(a*b)+cosf(a*b);
                }
            }
        };
        gettimeofday (&end, NULL);
        diff = (end.tv_sec*1000000+end.tv_usec) - (start.tv_sec*1000000+start.tv_usec);
        printf ("Nice-Value = %3d, Rechenzeit = %10d msec\n", niceval, diff);
        exit (0);
    };
    return;
}

int main () {
    nicetest (-19); nicetest (0); nicetest (10); nicetest (19);
    wait(NULL); wait(NULL); wait(NULL); wait(NULL);
    return 0;
}
```

```
esser@ubu64:nicetest$ ./a.out
Nice-Value = -19, Rechenzeit = 3009017 msec
Nice-Value = 0, Rechenzeit = 3070371 msec
Nice-Value = 10, Rechenzeit = 4569247 msec
Nice-Value = 19, Rechenzeit = 5797591 msec
esser@ubu64:nicetest$ sudo su
root@ubu64:/home/esser/nicetest# ./a.out
Nice-Value = -19, Rechenzeit = 1511453 msec
Nice-Value = 0, Rechenzeit = 3217840 msec
Nice-Value = 10, Rechenzeit = 4755353 msec
Nice-Value = 19, Rechenzeit = 6093009 msec
```

nice(): Priorität ändern (4)

- Wichtig: `nice()` interpretiert Argument als relative Änderung des aktuellen Nice-Werts (wie bei Shell-Kommando `nice`):
 - `nice(v)`: aktuellen Nice-Wert um `v` erhöhen
 - `nice -n v kommando`: Kommando mit Nice-Wert `NIC+v` (`NIC` = aktueller Nice-Wert in der Shell) ausführen
- nur `root` darf Priorität erhöhen; es ist also kein `nice(10); ... ; nice(-10)` möglich
- Sohnprozesse (`fork`) erben Nice-Wert des Vaters

Prozessgruppen und Sessions (2)

• Kontrollierendes Terminal:

- jeder Textmodus-Login (Textkonsolen, die Sie über Strg-Alt-F1 bis Strg-Alt-F6 erreichen)
- jedes Terminal-Fenster unter X (bzw. jeder Tab in einem Multi-Tab-Terminalprogramm)

```
root@ubu64:~/home/esser/tmp/forkwait# ps -C getty,bash,mc -o pid,sess,ttty,cmd
PID  SESS TT      CMD
709  709  tty4   /sbin/getty -8 38400 tty4
715  715  tty5   /sbin/getty -8 38400 tty5
724  724  tty2   /sbin/getty -8 38400 tty2
726  726  tty3   /sbin/getty -8 38400 tty3
730  730  tty6   /sbin/getty -8 38400 tty6
1505 1505 pts/0   bash
3434 3434 pts/1   bash
4945 3434 pts/1   bash
5941 1010 tty1    -bash
6064 1010 tty1    mc
6066 6066 pts/2   bash -rcfile .bashrc
```

Anmeldeprozesse auf den Textkonsolen 2-6 (tty2-tty6)

Bash-Prozesse in Terminalfenstern 0 und 1 (X)

Benutzer arbeitet auf tty1

Bash-Prozess in Fenster 2 (X)

Prozessgruppen und Sessions (1)

- Prozesse lassen sich zu Gruppen und diese zu Sessions zusammenfassen
- Alle Mitglieder einer Gruppe können gemeinsam signalisiert werden (z. B.: Abbruch)
- Sessions stehen oft für Terminal-Sitzungen; alle Prozesse in einer Session haben dasselbe „kontrollierende Terminal“ (TTY)

Prozessgruppen und Sessions (3)

- Neue Gruppe erzeugen: `setpgid(0,0)`
- Neue Session erzeugen: `setsid()`

erzeugen eine neue Gruppe bzw. Session, wobei als Gruppen-ID / Session-ID die Prozess-ID des aufrufenden Prozess verwendet wird

- Erzeugen einer neuen Session bewirkt immer auch das Erzeugen einer neuen Gruppe
- `setsid` funktioniert nicht, wenn der aufrufende Prozess „Prozessgruppenführer“ ist (PID = PGID)
- neue Session hat kein kontrollierendes Terminal (TTY)

Prozessgruppen und Sessions (4)

„setsid gibt -1 (für Fehler) zurück, wenn der aufrufende Prozess bereits ein Prozessgruppenführer ist. Um dies zu verhindern, kreiert man üblicherweise mittels fork einen Kindprozess, der weiterläuft, während sich der Elternprozess beendet. Der Kindprozess kann nämlich kein Prozessgruppenführer sein, da er zwar die Prozessgruppen-ID vom Elternprozess erbt, aber in jedem Fall eine neue Prozess-ID erhält, die niemals eine Prozessgruppen-ID sein kann, da sie neu ist.“

(Helmut Herold, Linux-Unix-Systemprogrammierung)

Abfragen von P.-Gruppe, Session

- Über die Funktionen

- getpgid (pid)
- getsid (pid)

lassen sich jederzeit die Gruppen- und Session-Zugehörigkeiten feststellen.

(für den aufrufenden Prozess mit pid=0)

- Prozessgruppenführer: PGID==PID
- Sessionführer: SID==PID

Prozessgruppen und Sessions (5)

```
int main () {
    fork ();
    setsid (); // new session (works only in child)
    fork ();
    setpgid (0,0); // new group
    fork ();
    sleep(5);
    return 0;
}
```

```
esser@ubu64:~$ ps -C setpgid -o pid,ppid,pgid,sid,ttty,cmd; pstree -p | cut -c34- | grep setpgid
```

```
  PID PPID PGID  SID TT  CMD
5757 4945 5757 3434 pts/1 ./setpgid
5758 5757 5758 5758 ? ./setpgid
5759 5757 5759 3434 pts/1 ./setpgid
5760 5757 5757 3434 pts/1 ./setpgid
5761 5759 5759 3434 pts/1 ./setpgid
5762 5758 5762 5758 ? ./setpgid
5763 5758 5758 5758 ? ./setpgid
5764 5762 5762 5758 ? ./setpgid
```

```
Session 3434: 5757, 5759, 5760, 5761
Session 5758: 5758, 5762, 5763, 5764
Vier Prozessgruppen: 5757, 5758, 5759, 5762
```

```
bash (3434) --su (4937) --bash (4945) --setpgid(5757) --setpgid(5758) --setpgid(5762) --setpgid(5764)
|
|   --setpgid(5763)
|   |
|   |   --setpgid(5759) --setpgid(5761)
|   |   |
|   |   |   --setpgid(5760)
```

Informationen über Prozesse (1)

- /proc enthält für jeden Prozess einen Ordner PID (also /proc/1, /proc/2, ...)
- In jedem dieser Ordner gibt es zahlreiche Dateien und Verzeichnisse mit Informationen über den Prozess
 - status: (mensch-)lesbare Statusinformationen
 - stat: maschinenlesbare Statusinformationen
 - cmdline: Kommando
 - environ: Umgebungsvariablen
 - fd, fdinfo: Informationen zu offenen Dateien

Informationen über Prozesse (2)

```
# cat /proc/7195/envIRON ; echo
SHELL=/bin/bashTERM=xtermXDG_SESSION_COOKIE=c420f37852122ff2d7e4eb894f8e
b0a2-1335107306.31018-208107520USER=rootSUDO_USER=esserSUDO_UID=1000USER
NAME=rootMAIL=/var/mail/rootPATH=/usr/local/sbin:/usr/local/bin:/usr/sbi
n:/usr/bin:/sbin:/bin:/usr/gamesPWD=/home/esser/tmp/forkwaitLANG=de_DE.U
TF-8SHLVL=1SUDO_COMMAND=/bin/suHOME=/rootLOGNAME=rootLESSOPEN=| /usr/bin
/lesspipe %sSUDO_GID=1000DISPLAY=:0LESSCLOSE=/usr/bin/lesspipe %s %sCO
LORTERM=gnome-terminalXAUTHORITY=/var/run/gdm/auth-for-esser-Zt6Xhr/data
base_=/usr/bin/ncedit

# tr '\0' '\n' < /proc/7195/envIRON
SHELL=/bin/bash
TERM=xterm
XDG_SESSION_COOKIE=c420f37852122ff2d7e4eb894f8eb0a2-1335107306.31018-
208107520
USER=root
SUDO_USER=esser
SUDO_UID=1000
USER_NAME=root
MAIL=/var/mail/root
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/g
ames
PWD=/home/esser/tmp/forkwait
LANG=de_DE.UTF-8
[...]
```

Informationen über Prozesse (4)

```
# ps -C vi -o pid,cmd
PID CMD
9368 vi forkexec.c

# tr '\0' ' ' < /proc/9368/cmdline ; echo
vi forkexec.c

# ls -l /proc/9368/fd/
insgesamt 0
lrwx----- 1 esser esser 64 2012-04-24 01:07 0 -> /dev/pts/0
lrwx----- 1 esser esser 64 2012-04-24 01:07 1 -> /dev/pts/0
lrwx----- 1 esser esser 64 2012-04-24 01:07 2 -> /dev/pts/0
lrwx----- 1 esser esser 64 2012-04-24 01:07 4 ->
/home/esser/tmp/uebung02/.forkexec.c.swp

# cat /proc/9368/fdinfo/4
pos: 12288
flags: 0100002
```

Informationen über Prozesse (3)

```
# cat /proc/7195/status
Name: nedit
State: S (sleeping)
Tgid: 7195
Pid: 7195
PPid: 1
TracerPid: 0
Uid: 0 0 0 0
Gid: 0 0 0 0
FDSize: 256
Groups: 0
VmPeak: 61472 kB
VmSize: 61468 kB
VmLck: 0 kB
VmHWM: 7224 kB
VmRSS: 7224 kB
VmData: 4764 kB
VmStk: 88 kB
VmExe: 1188 kB
VmLib: 6464 kB
VmPTE: 140 kB
Threads: 1
SigQ: 2/16382

SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000000000
SigIgn: 0000000000000000
SigCgt: 0000000000000000
CapInh: 0000000000000000
CapPrm: ffffffff
CapEff: ffffffff
CapBnd: ffffffff
Cpus_allowed: 1
Cpus_allowed_list: 0
Mems_allowed: 00000000,00000001
Mems_allowed_list: 0
voluntary_ctxt_switches: 13609
nonvoluntary_ctxt_switches: 620

# cat /proc/7195/stat
7195 (nedit) S 1 7195 3434 34817 9334
4202752 2021 0 19 0 96 51 0 0 20 0 1 0
3941746 62943232 1806
18446744073709551615 4194304 5410572
140734327978864 140734327968016
140314005931272 0 0 0 0
18446744071580244841 0 0 17 0 0 0 0 0 0
```

Informationen über Prozesse (5)

```
Auszug aus
/usr/include/asm-generic/fcntl.h:

#define O_RDONLY 00000000
#define O_WRONLY 00000001
#define O_RDWR 00000002
#define O_CREAT 00000100
#define O_EXCL 00000200
#define O_NOCTTY 00000400
#define O_TRUNC 00001000
#define O_APPEND 00002000
#define O_NONBLOCK 00004000
#define O_SYNC 00010000
#define FASYNC 00020000 // BSD comp.
#define O_DIRECT 00040000
#define O_LARGEFILE 00100000
#define O_DIRECTORY 00200000
#define O_NOFOLLOW 00400000
#define O_NOATIME 01000000
#define O_CLOEXEC 02000000

# cat /proc/9368/fdinfo/4
pos: 12288
flags: 0100002

0100002 =
0100000 (O_LARGEFILE)
+0000002 (O_RDWR)
```

argc und argv (1)

- Unix-Tools werten meist Argumente aus
- Deklariere main() als

```
int main (int argc, char *argv[])
```
- argc: Anzahl der Argumente (Programmname = 1. Argument)
- *argv[]: Array mit Argument-Strings

argc und argv (2)

```
#include <stdio.h> // printf
#include <stdlib.h> // exit

int main (int argc, char *argv[]) {
    int i;
    printf ("argc = %d\n", argc);
    for (i=0 ; i<argc ; i++) {
        printf ("argv[%d] = %s\n", i, argv[i]);
    };
    exit (0);
}
```

```
esser@ubu64:~/argc$ ./argumente
argc = 1
argv[0] = ./argumente
esser@ubu64:~/argc$ ./argumente eins zwei
argc = 3
argv[0] = ./argumente
argv[1] = eins
argv[2] = zwei
```

argc und argv (3)

- vereinfachte Auswertung der Argumente mit getopt und getopt_long
- getopt verarbeitet Kurzoptionen (-a, -b) und deren Kombinationen (-ab)
- getopt_long verarbeitet auch Langoptionen (--longoption)

argc und argv (4)

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main (int argc, char **argv) {
    int aflag = 0; int bflag = 0;
    char *cvalue = NULL; int index; int c;

    opterr = 0;

    while ((c = getopt (argc, argv, "abc:")) != -1)
        switch (c) {
            case 'a':
                aflag = 1; break;
            case 'b':
                bflag = 1; break;
            case 'c':
                cvalue = optarg; break;
            case '?':
                if (optopt == 'c')
                    fprintf (stderr, "Option -%c requires an argument.\n", optopt);
                else if (isprint (optopt))
                    fprintf (stderr, "Unknown option `-%c'.\n", optopt);
                else
                    fprintf (stderr, "Unknown option character `\\x%x'.\n", optopt);
                return 1;
            default:
                abort ();
        }

    printf ("aflag = %d, bflag = %d, cvalue = %s\n", aflag, bflag, cvalue);

    for (index = optind; index < argc; index++)
        printf ("Non-option argument %s\n", argv[index]);
    return 0;
}
```

argc und argv (5)

```
# getopt-test
aflag = 0, bflag = 0, cvalue = (null)

# getopt-test -a
aflag = 1, bflag = 0, cvalue = (null)

# getopt-test -ab
aflag = 1, bflag = 1, cvalue = (null)

# getopt-test -c test
aflag = 0, bflag = 0, cvalue = test

# getopt-test argument
aflag = 0, bflag = 0, cvalue = (null)
Non-option argument argument

# getopt-test -b argument -c test -a
aflag = 1, bflag = 1, cvalue = test
Non-option argument argument

# getopt-test -b argument -c test -a mehr
aflag = 1, bflag = 1, cvalue = test
Non-option argument argument
Non-option argument mehr

# getopt-test -f
Unknown option '-f'.
```

argc und argv (6)

Erklärungen zum Beispiel-Programm:

- drittes getopt-Argument gibt zulässige Optionen an, im Beispiel: "abc : "
- Option -c erwartet ein Argument (Postfix :)
- optopt enthält letzte Option, wenn auf :-Option kein passendes Argument folgt
- opterr = 0 unterdrückt getopt-eigene Fehlerausgabe bei unbekannter Option