

```

Sep 19 14:20:41 amd64 sshd[10474]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64557
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[6609]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd[6694]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 20 15:27:35 amd64 sshd[9071]: 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[17878]: (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[3188]: 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 17:43:26 amd64 sshd[31269]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
Sep 21 18:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 18:43:26 amd64 /usr/sbin/cron[4674]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 22 01:00:01 amd64 /usr/sbin/cron[5689]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 22 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[5689]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 22 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 01:00:01 amd64 /usr/sbin/cron[5689]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 23 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 02:00:01 amd64 /usr/sbin/cron[5689]: (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:192.168.1.15 port 59771 ssh2
Sep 23 18:04:05 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:34 amd64 sshd[6641]: 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[1837]: 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: amd_seq_mid1_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 kernel: amd_seq_oss: 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[662]: (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[1484]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 25 02:00:02 amd64 /usr/sbin/cron[662]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
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[9211]: 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[1154]: 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[11686]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd[11608]: 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

```

Interrupts (2/2)

Interrupt Handler (2)

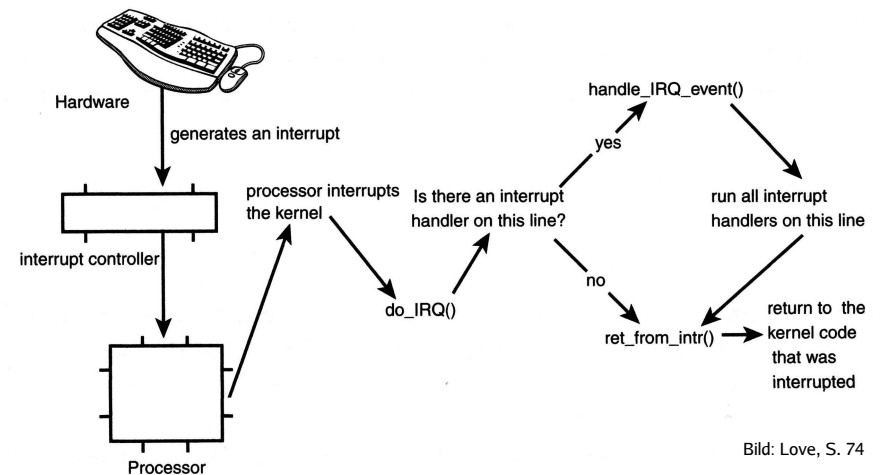


Bild: Love, S. 74

Interrupt Handler (1)

Treiber registrieren mit Interrupt handler:

```

int request_irq(
    unsigned int irq,           /* Welche IRQ-Nummer?          */
    irqreturn_t (*handler)(int, void *, struct pt_regs *),
    unsigned long irqflags,
    const char * devname,     /* Gerätename->/proc/int.* */
    void *dev_id);

```

- Interrupt mit IRQ `irq` wird ausgelöst
- BS ruft Interrupt handler `handler()` auf
- Flags:
 - SA_SHIRQ: Interrupt für mehrere Treiber
 - SA_INTERRUPT: Lokale Interrupts werden gesperrt
 - SA_SAMPLE_RANDOM: Interrupts treten „zufällig“ auf, nutzen: Entropie-Vergrößerung, Zufallszahlen

Interrupt Handler (3)

Beispiel: Timer, RTC-Chip auf dem Mainboard

`drivers/char/rtc.c, rtc_init()`

```

if (request_irq(
    RTC_IRQ,           /* RTC Interrupt: 8          */
    rtc_interrupt,    /* Interrupt handler         */
    SA_INTERRUPT,     /* Flag: lokale Int. sperren */
    "rtc",           /* Gerätename "rtc"         */
    (void *)&rtc_port) /* ID                        */
)
{
    printk(KERN_ERR "rtc: cannot
register IRQ %d\n", RTC_IRQ);
    return -EIO;      /* EIO: I/O Error          */
}

```

Interrupt Handler (4)

```
irqreturn_t rtc_interrupt(int irq, void *dev_id, struct pt_regs *regs) {
    spin_lock (&rtc_lock);
    rtc_irq_data += 0x100;          /* globale Variable! */
    rtc_irq_data &= ~0xff;
    rtc_irq_data |= (CMOS_READ(RTC_INTR_FLAGS) & 0xF0);

    if (rtc_status & RTC_TIMER_ON)
        mod_timer(&rtc_irq_timer, jiffies + HZ/rtc_freq + 2*HZ/100);

    spin_unlock (&rtc_lock);

    /* Now do the rest of the actions */
    spin_lock(&rtc_task_lock);
    if (rtc_callback)
        rtc_callback->func(rtc_callback->private_data);
    spin_unlock(&rtc_task_lock);
    wake_up_interruptible(&rtc_wait);

    kill_fasync (&rtc_async_queue, SIGIO, POLL_IN);

    return IRQ_HANDLED;
}
```

Interrupt Handler (6)

Eigener Handler wird aufgerufen in

kernel/irq/handle.c, handle_IRQ_event():

```
int handle_IRQ_event(unsigned int irq, struct pt_regs *regs,
                    struct irqaction *action) {
    int ret, retval = 0, status = 0;

    if (!(action->flags & SA_INTERRUPT))
        local_irq_enable();

    do {
        ret = action->handler(irq, action->dev_id, regs);
        if (ret == IRQ_HANDLED)
            status |= action->flags;
        retval |= ret;
        action = action->next;
    } while (action);

    if (status & SA_SAMPLE_RANDOM)
        add_interrupt_randomness(irq);
    local_irq_disable();

    return retval;
}
```

Interrupt Handler (5)

RTC: I/O-Adressen

- 0x70 (lesen)
- 0x71 (schreiben)

```
/usr/include/linux/mc146818rtc.h:
extern spinlock_t rtc_lock; /* serialize CMOS RAM access */

#define RTC_PORT(x) (0x70 + (x))

#define CMOS_READ(addr) ( {
    outb_p((addr), RTC_PORT(0));
    inb_p(RTC_PORT(1));
} )
```

Interrupt Handler (7)

Zugriff auf Treiber-Daten aus Programmen

```
ssize_t rtc_read(struct file file, char *buf, size_t count, loff_t *ppos) {
    DECLARE_WAITQUEUE(wait, current);
    unsigned long data;
    ssize_t retval;

    add_wait_queue(&rtc_wait, &wait);
    current->state = TASK_INTERRUPTIBLE;
    do {
        spin_lock_irq(&rtc_lock);
        data = rtc_irq_data; /* globale Variable; auch */
        rtc_irq_data = 0; /* in der Service-Routine! */
        spin_unlock_irq(&rtc_lock);

        if (data != 0) break;
        [.....]
        schedule(); /* schlafen legen */
    } while(1);
    retval = put_user(data, (unsigned long *)buf);
    [.....]

    current->state = TASK_RUNNING;
    remove_wait_queue(&rtc_wait, &wait);
    return retval;
}
```

Interrupt Handler (8)

Wichtig: In welchem Context läuft was?

- **User Context:** unterbrechbar (HW oder SW interrupts), kann system calls aufrufen,
- **Process Context:** nach Software Interrupt aus User Context, läuft im Kernel, Daten zwischen Kernel- und Prozessspeicher übertragen, nur durch HW-Interrupt unterbrechbar
- **Kernel Context:** Funktionen des Kernels, kein Datenaustausch zwischen Kernel- und User-Space, nur durch HW-Interrupt unterbrechbar
- **Interrupt Context:** Software- und Hardware-Interrupts

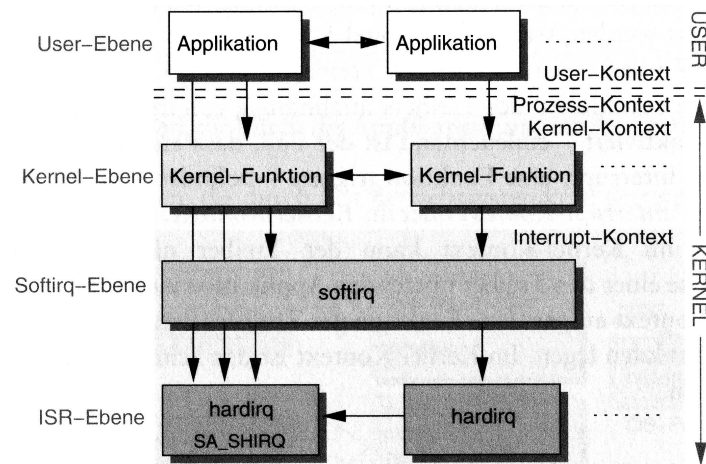
Interrupt Handler (10)

Top und bottom half / Tasklet

Bottom half heißt im Linux-Kernel (seit Version 2.6) **Tasklet**

- Interrupt Service Routine (top half) erledigt das Wichtigste (zeitkritische Dinge), erzeugt Tasklet und beendet sich – dabei sind Interrupts gesperrt
- Tasklets führen längere Berechnungen durch, die zur Interrupt-Verarbeitung gehören – dabei sind Interrupts zugelassen

Interrupt Handler (9)



a → b
a kann durch b unterbrochen werden

Bild: Quade/Kunst, S. 20

Interrupt Handler (10)

Tasklets

- Tasklet ist kein Prozess (struct tasklet_struct), läuft direkt im Kernel; im Interrupt-Context
- Zwei Prioritäten:
 - *tasklet_hi_schedule*: startet direkt nach ISR
 - *tasklet_schedule*: startet erst, wenn kein anderer Soft IRQ mehr anliegt

Interrupt Handler (11)

Mehr Informationen:

- [1] Linux Kernel 2.4 Internals, Kapitel 2,
http://www.faqs.org/docs/kernel_2_4/ki-2.html
- [2] J. Quade, E.-K. Kunst: „Linux-Treiber entwickeln“,
dpunkt-Verlag,
<http://ezs.kr.hsnr.de/TreiberBuch/html/>

System Calls (2)

Software Interrupts

- Argumente über Register übergeben
- Maschinenbefehl `int 0x80` ausführen
→ Trap (Software Interrupt), Wechsel in Kernel Mode
- Funktion `system_call` in `arch/i386/kernel/entry.S` ausführen
- Dort: `call sys_call_table+4*(syscall_number from %eax)`
→ Sprung in C-Routinen, `sys_*` (`sys_open`, `sys_exit` etc.)
- Syscall-Tabelle definiert in `arch/i386/kernel/syscall_table.S`

System Calls (1)

`asm/unistd.h`: Über 300 System Calls

```
/*
 * This file contains the system call
 * numbers.
 */
#define __NR_restart_syscall 0
#define __NR_exit 1
#define __NR_fork 2
#define __NR_read 3
#define __NR_write 4
#define __NR_open 5
#define __NR_close 6
#define __NR_waitpid 7
#define __NR_creat 8
#define __NR_link 9
#define __NR_unlink 10
#define __NR_execve 11
#define __NR_chdir 12
#define __NR_time 13
#define __NR_mknod 14
#define __NR_chmod 15
#define __NR_lchown 16

#define __NR_break 17
#define __NR_oldstat 18
#define __NR_lseek 19
#define __NR_getpid 20
#define __NR_mount 21
#define __NR_umount 22
#define __NR_setuid 23
#define __NR_getuid 24
#define __NR_stime 25
#define __NR_ptrace 26
#define __NR_alarm 27
#define __NR_oldfstat 28
#define __NR_pause 29
#define __NR_utime 30
#define __NR_stty 31
#define __NR_gtty 32
#define __NR_access 33
#define __NR_nice 34
#define __NR_ftime 35
#define __NR_sync 36
#define __NR_kill 37
...
```

In `fs/open.c`: System Calls (3)

```
long do_sys_open(int dfd, const char __user *filename, int flags, int mode)
{
    char *tmp = getname(filename);
    int fd = PTR_ERR(tmp);

    if (!IS_ERR(tmp)) {
        fd = get_unused_fd();
        if (fd >= 0) {
            struct file *f = do_filp_open(dfd, tmp, flags, mode);
            if (IS_ERR(f)) {
                put_unused_fd(fd);
                fd = PTR_ERR(f);
            } else {
                fsnotify_open(f->f_dentry);
                fd_install(fd, f);
            }
            putname(tmp);
        }
        return fd;
    }

    asmlinkage long sys_open(const char __user *filename, int flags, int mode)
    {
        long ret;

        if (force_o_largefile())
            flags |= O_LARGEFILE;

        ret = do_sys_open(AT_FDCWD, filename, flags, mode);
        /* avoid REGPARM breakage on x86: */
        prevent_tail_call(ret);
        return ret;
    }
}
```

System Calls (4)

Beispiel für einen System Call:

Library-Funktion `fread()`

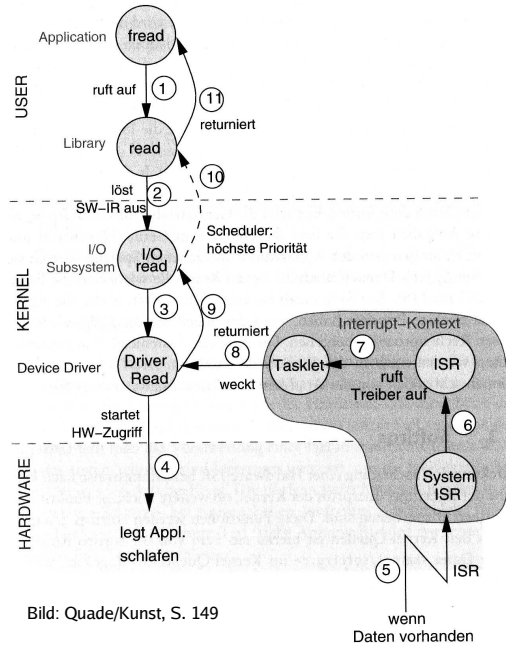


Bild: Quade/Kunst, S. 149

3. Interrupts (2/2) – Folie 17

Bibliotheksfunktionen

open(): Datei zum Lesen/Schreiben öffnen

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
int creat(const char *pathname, mode_t mode);
```

Rückgabewert: File Descriptor

```
man 2 open
```

Beispiel:

```
fd = open("/tmp/datei.txt", O_RDONLY);
```

```
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[50323]: (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 12:48:41 amd64 sshd[6609]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd[6694]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 20 15:27:35 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[17878]: (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 64456
Sep 21 17:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:53:39 amd64 sshd[31269]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
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[41111]: (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[51331]: (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[25555]: (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[6541]: 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[6606]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61893
Sep 24 01:00:01 amd64 /usr/sbin/cron[11111]: (root) CMD (/sbin/evlogmgr -c "age > *30d*")
Sep 24 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron[11111]: (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[21297]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62709
Sep 24 13:49:08 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_midi_0: snd_seq_midi_0: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_oss: 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[6621]: (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[1484]: (root) CMD (/sbin/evlogmgr -c "age > *30d*")
Sep 25 02:00:02 amd64 /usr/sbin/cron[1484]: (root) CMD (/sbin/evlogmgr -c "age > *30d*")
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[11608]: 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
```

System Calls für Programmierer: Standardfunktionen in C und Python

Bibliotheksfunktionen

fopen(): Datei zum Lesen/Schreiben öffnen

```
FILE *fopen(const char *path, const char *mode);
```

mode: r = read, w = write (truncate), a = write (append), r+ = read/write

Rückgabewert: File Pointer (nicht Descriptor!)

```
man fopen
```

Beispiel:

```
fp = fopen("/tmp/datei.txt", "r");
```

Bibliotheksfunktionen

read(): Daten aus Datei (File Descriptor) lesen

```
ssize_t read(int fd, void *buf, size_t count);
```

Rückgabewert: Anzahl gelesene Bytes

man 2 read

Beispiel:

```
int bufsiz=128;
char line[bufsiz];
int fd = open( "/etc/fstab", O_RDONLY );
int len;
while ( len = read ( fd, line, bufsiz ) > 0 ) {
    printf ( line );
}
close(fd);
exit(0);
}
```

Bibliotheksfunktionen

write(): Daten in Datei (File Descriptor) schreiben

```
ssize_t write(int fd, void *buf, size_t count);
```

Rückgabewert: Anzahl geschriebene Bytes

man 2 write

Beispiel:

```
main() {
    char message[] = "Hello world\n";
    int fd = open( "/tmp/datei.txt",
                  O_CREAT | O_WRONLY, S_IRUSR | S_IWUSR );
    write ( fd, message, sizeof(message) );
    perror();
    close(fd);
    exit(0);
}
```

Bibliotheksfunktionen

fread(): Daten aus Datei (File Pointer) lesen

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

Rückgabewert: Anzahl gelesene Blöcke (nicht Bytes)

man fread

Beispiel:

```
int bufsiz=128; int len;
char line[bufsiz]; FILE *fp;
fp = fopen( "/etc/fstab", "r" );
while ( !feof(fp) ) {
    if (fread ( line, bufsiz, 1, fp ) > 0) {
        printf ( line );
    }
}
close(fp);
printf("\n");
```

Bibliotheksfunktionen

fwrite(): Daten in Datei (File Pointer) schreiben

```
size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

Rückgabewert: Anzahl geschriebene Blöcke (nicht Bytes)

man fwrite

Beispiel:

```
main () {
    char message[] = "Hello world!\n";
    FILE *fp;

    fp = fopen( "/tmp/datei.txt", "w" );
    fwrite ( message, sizeof(message), 1, fp );
    close(fp);
    exit(0);
}
```

Bibliotheksfunktionen

close(): Datei (File Descriptor) schließen

```
int close(int fd);
```

Rückgabewert: 0 bei Erfolg, sonst -1 (`errno` enthält dann Grund)

```
man 2 close
```

Beispiel:

```
close(fd);
```

Bibliotheksfunktionen

Python: open, readlines, write, close

Beispiel: Datei zeilenweise kopieren

```
#!/usr/bin/python
fd = open("/etc/fstab", "r")
lines = fd.readlines()
fd.close()
```

```
fd = open("/tmp/datei.txt", "w")
for l in lines:
    fd.write(l)
fd.close()
```

Bibliotheksfunktionen

fclose(): Datei (File Pointer) schließen

```
int fclose(FILE *fp);
```

Rückgabewert: 0 im Erfolgsfall, sonst EOF (und `errno` enthält den Fehlergrund)

```
man fclose
```

Beispiel:

```
fclose(fp);
```

Bibliotheksfunktionen

exit(): Programm beenden

```
void exit(int status);
```

Kein Rückgabewert, aber status wird an aufrufenden Prozess weitergegeben.

```
man 3 exit
```

Beispiel:

```
exit(0);
```

Bibliotheksfunktionen

fork(): Neuen Prozess starten

```
pid_t fork(void);
```

Rückgabewert: Child-PID (im Vaterprozess); 0 (im Sohnprozess); -1 (im Fehlerfall)

man fork

Beispiel:
pid=fork()

Bibliotheksfunktionen

Python: Programm starten wie in C

Vorsicht: Beendet den Python-Interpreter!

```
import os
os.execl("/usr/bin/vi", "", "/etc/fstab")
```

Bibliotheksfunktionen

exec(): Anderes Programm in Prozess laden

```
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execl(const char *path, const char *arg, ..., char * const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
```

Rückgabewert: keiner (Funktion kehrt nicht zurück)
Parameter arg0 (Name), arg1, ...; letztes Argument: NULL-Zeiger

man 3 exec

Beispiele:
execl("/usr/bin/vi", "", "/etc/fstab", (char *) NULL);
execlp("vi", "", "/etc/fstab", (char *) NULL);

Bibliotheksfunktionen

Python: Programm starten und danach Python-Skript fortsetzen

```
import os
os.system("vi /etc/fstab")
```

Programmausgabe weiter verarbeiten (Pipe)

```
output=os.popen("cat /etc/fstab").read()
print output
```


Bibliotheksfunktionen

Python: fork() und wait() wie in C-Programmen

fork() startet 2. Python-Prozess und führt darin das gleiche Programm aus

```
#!/usr/bin/python
import os
import time
pid=os.fork()
if pid==0:
    time.sleep(5)
    print ">> Ich bin der Sohn."
else:
    print "Ich bin der Vater. Mein Sohn hat die PID ",pid
    print "Ich warte jetzt auf den Sohn..."
    os.wait()
    print "Er ist fertig."
```