```
Sep 19 14:20:18 amd64 sshd[20494]: Accepted [illegible]er from ::ffff:87.234.201.207 port 61557
Sep 19 14:27:41 amd64 syslog-ng[7653]: ST[illegible]
Sep 20 01:00:01 amd64 /usr/sbin/cron[2[illegible]/sbin/evlogmgr -c "severity=DEBUG")
Sep 20 01:00:01 amd64 syslog-ng[765[illegible]
Sep 20 02:00:01 amd64 /usr/sbin/c[illegible]in/evlogmgr -c 'age > "30d"')
Sep 20 02:00:01 amd64 syslog-n[illegible]
Sep 20 12:46:44 amd64 sshd[[illegible]:ffff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 sy[illegible]
Sep 20 12:48:41 amd64 [illegible]:ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd[illegible]m ::ffff:87.234.201.207 port 62514
Sep 20 15:27:35 [illegible]rom ::ffff:87.234.201.207 port 64242
Sep 20 15:27:[illegible]
Sep 20 16:[illegible]esser from ::ffff:87.234.201.207 port 63375
Sep 20 1[illegible]pped 0
Sep 2[illegible]a for esser from ::ffff:87.234.201.207 port 63546
Se[illegible] (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
[illegible]ATS: dropped 0
[illegible]878]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
[illegible]]: STATS: dropped 0
[illegible]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63397
[illegible]g[7653]: STATS: dropped 0
[illegible]1269]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
[illegible]slog-ng[7653]: STATS: dropped 0
[illegible]syslog-ng[7653]: STATS: dropped 0
[illegible]4 /usr/sbin/cron[4674]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
[illegible]amd64 syslog-ng[7653]: STATS: dropped 0
[illegible]1 amd64 /usr/sbin/cron[5499]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
S[illegible]0:01 amd64 syslog-ng[7[illegible]3]: STATS: d[illegible]pped 0
Sep[illegible]0:23:21 amd64 syslog-ng[7[illegible]
Sep [illegible] 01:00:01 amd64 /usr/sbin/c[illegible]
Sep 23 01:00:01 amd64 syslog-ng[7[illegible]
Sep 23 02:00:01 amd64 /usr/sbin/c[illegible]
Sep 23 02:00:01 amd64 syslog-ng[7[illegible] dropped 0
Sep 23 18:04:05 amd64 sshd[6554]: Accepted publickey for esser from ::ffff:192.168[illegible]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 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 15:42:07 amd64 kernel: snd_seq_oss: unsupported module, tainting kernel.
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 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[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
```

English Edition

# Interrupts (2/2)

## Interrupt Handler (1)

### Register drivers with interrupt handler:

```
int request_irq(
  unsigned int irq,                /* Which IRQ number?      */
  irqreturn_t (*handler)(int, void *, struct pt_regs *),
  unsigned long irqflags,
  const char * devname,            /* devicename->/proc/int..*/
  void *dev_id);
```

- Interrupt with IRQ number `irq` occurs
- OS calls interrupt handler `handler()`
- Flags:

| | |
|---|---|
| SA_SHIRQ: | Interrupt for several drivers |
| SA_INTERRUPT: | Disable local interrupts |
| SA_SAMPLE_RANDOM: | Interrupts are "random" – use that: increase entropy, for random numbers |

## Interrupt Handler (2)

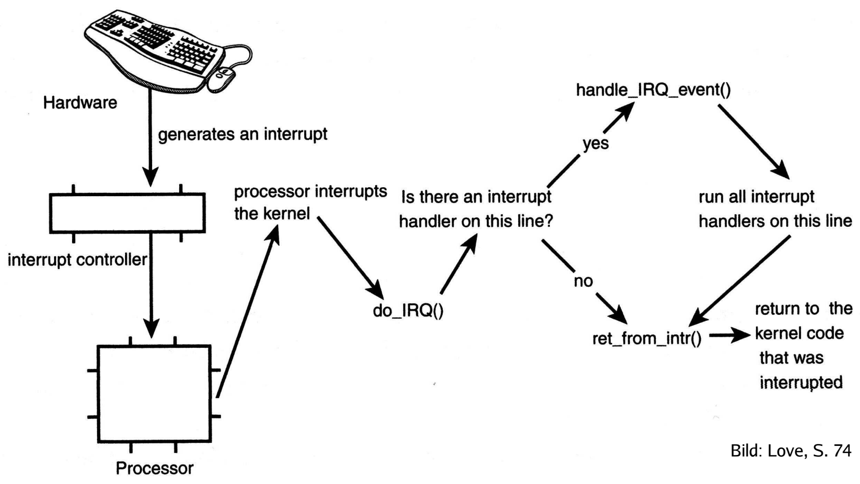
Hardware
generates an interrupt
interrupt controller
Processor
processor interrupts the kernel
do_IRQ()
Is there an interrupt handler on this line?
yes
handle_IRQ_event()
run all interrupt handlers on this line
no
ret_from_intr()
return to the kernel code that was interrupted

Bild: Love, S. 74

## Interrupt Handler (3)

### Example: Timer, RTC chip on mainboard

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

```
if (request_irq(
       RTC_IRQ,               /* RTC Interrupt: 8          */
       rtc_interrupt,         /* Interrupt handler         */
       SA_INTERRUPT,          /* Flag: disable local int.  */
       "rtc",                 /* device name "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;          /* global 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 (5)

RTC: I/O addresses
- 0x70 (read)
- 0x71 (write)

```
/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 (6)

handler will be called 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 (7)

How programs can access driver data

```
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;            /* global variable; also in   */
                rtc_irq_data = 0;               /* interrupt service routine! */
                spin_unlock_irq(&rtc_lock);

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

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

## Interrupt Handler (8)

**Important: What runs in what context?**

- **User Context:** interruptible (HW or SW interrupts), can issue system calls
- **Process Context:** entered after software interrupt (from user context), runs in the kernel, transfer data between Kernel and process spaces, only interruptible by HW interrupts
- **Kernel Context:** functions of the kernel, no data transfer between Kernel and user space, only interruptible by HW interrupt
- **Interrupt Context:** software and hardware interrupts

## Interrupt Handler (9)

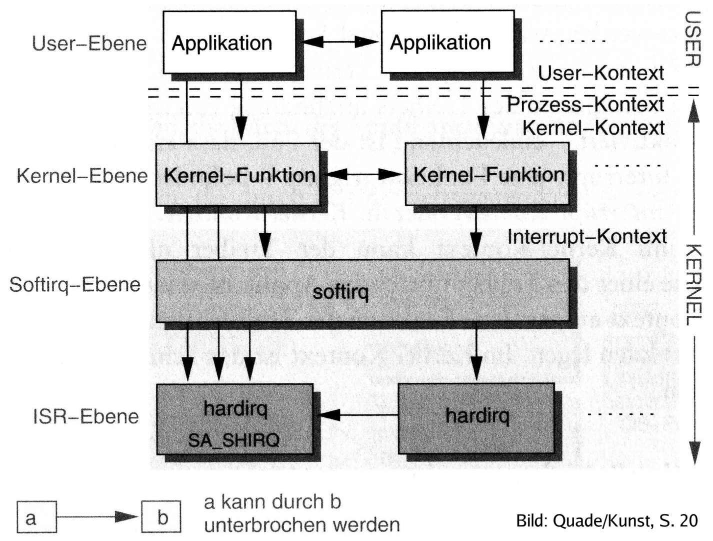

Bild: Quade/Kunst, S. 20

## Interrupt Handler (10)

**Top and bottom half / Tasklet**

**Bottom half** was renamed in Linux-Kernel (since version 2.6) as **Tasklet**

- Interrupt Service Routine (top half) handles the most important (time critical) tasks, generates tasklet and terminates – meanwhile interrupts disabled
- Tasklets run longer calculations which are part of processing the interrupt – with interrupts enabled

## Interrupt Handler (10)

**Tasklets**

- Tasklet is not a process (struct tasklet_struct), runs directly in Kernel; in Interrupt Context
- Two priorities:
  – *tasklet_hi_schedule*: starts immediately after ISR
  – *tasklet_schedule*: only starts when there are no more soft IRQs

## Interrupt Handler (11)

### More Information:

[1] Linux Kernel 2.4 Internals, Kapitel 2,
http://www.faqs.org/docs/kernel_2_4/lki-2.html

[2] J. Quade, E.-K. Kunst: „Linux-Treiber entwickeln“, dpunkt-Verlag,
http://ezs.kr.hsnr.de/TreiberBuch/html/

## System Calls (1)

*asm/unistd.h*: More than 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
...
```

## System Calls (2)

### Software Interrupts

- Put arguments into registers
- Execute machine instruction **`int 0x80`**
  ➞ Trap (Software Interrupt), switch to Kernel mode
- Execute function `system_call` in *arch/i386/kernel/entry.S*
- There: call sys_call_table+4*(syscall_number from %eax)
  ➞ Jump into C routines, `sys_*` (`sys_open`, `sys_exit` etc.)
- Syscall table defined in *arch/i386/kernel/syscall_table.S*

## System Calls (3)

In *fs/open.c*:

```
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)

Example for a system call:

library function *fread( )*

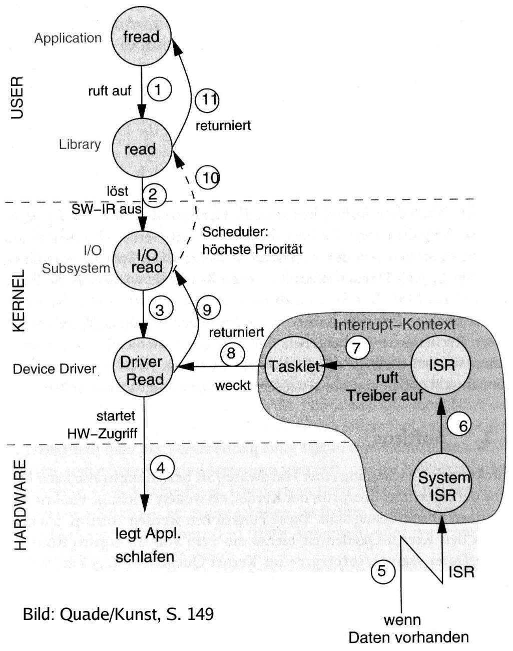

Bild: Quade/Kunst, S. 149

## Library Functions

**open():** open file for read/write

```
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);
```

Return value: File Descriptor

```
man 2 open
```

Example:
```
fd = open("/tmp/file.txt",O_RDONLY);
```

## System Calls for programmers:

## standard functions in C and Python

```
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[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 63397
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 [illegible]
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[4674]: (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[5499]: (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[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[6606]: 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[[illegible]]: (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 [illegible]
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 15:42:07 amd64 kernel: snd_seq_oss: unsupported module, tainting kernel.
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 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[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
```

## Library Functions

**fopen():** Open file for read/write

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

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

Return value: File Pointer (not descriptor!)

```
man fopen
```

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

## Library Functions

**read():** Read from file (file descriptor)

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

Return value: number of bytes read

```
man 2 read
```

Example:

```
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);
return 0;
}
```

bad C code...
see next slide

## More about read/open

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

int main (void) {
  int len;
  int bufsiz=128;
  char line[bufsiz+1];
  line[bufsiz] = '\0';

  int fd = open( "/etc/fstab", O_RDONLY );
  while ( (len = read ( fd, line, bufsiz )) > 0 ) {
    if ( len < bufsiz) { line[len]='\0'; }
    printf ("%s", line );
  }
  close(fd);
  return 0;
}
```

The local C guru looked at my C programs... Thanks go to Mirko Dölle!

## Library Functions

**fread():** Read from file (file pointer)

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

Return value: number of blocks read (not 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");
```

bad C code...
see next slide

## More about fread/fopen

```
#include <stdio.h>

int main (void) {

  int bufsiz=512;
  char line[bufsiz+1];
  line[bufsiz] = '\0';
  FILE *fp;

  fp = fopen( "/etc/fstab", "r" );
  int len;
  while ( !feof(fp) ) {
    if (fread ( line, bufsiz, 1, fp ) > 0) {
      if ( len < bufsiz) { line[len]='\0'; }
      printf ( "%s", line );
    }
  }
  fclose(fp);
  return 0;
}
```

The local C guru looked at my C programs... Thanks go to Mirko Dölle!

## Library Functions

**write():** write to file (file descriptor)

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

Return value: Number of bytes written

```
man 2 write
```

Example:
```
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);
  return 0;
}
```

## Library Functions

**fwrite():** write to file (file pointer)

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

Return value: number of blocks written (not bytes)

```
man fwrite
```

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

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

## Library Functions

**close():** close file (file descriptor)

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

Return value: 0 if successful, otherwise -1
(`errno` contains reason)

```
man 2 close
```

Example:
```
close(fd);
```

## Library Functions

**fclose():** close file (file pointer)

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

Return value: 0 if successful, otherwise EOF
(`errno` contains reason)

```
man fclose
```

Example:
```
close(fp);
```

## Library Functions

**Python:** open, readlines, write, close

Example: Copy file line by line

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

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

## Library Functions

**exit():** leave program

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

No return value, but exit status is returned to father process.

```
man 3 exit
```

Example:
`exit(0);`

## Library Functions

**fork():** Create a new process

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

Return value: Child-PID (in parent process);
0 (in child process); -1 (in case of error while trying to fork)

```
man fork
```

Example:
`pid=fork()`

## Library Functions

**exec():** Load different program in process

```
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(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[]);
```

Return value: none (function does not return)
Parameters arg0 (program name), arg1, ...; last argument: NULL pointer

```
man 3 exec
```

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

## Library Functions

**Python:** Launch program just like in C

Warning: terminates Python interpreter

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

## Library Functions

**Python:** launch program and continue with Python script afterwards

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

process program output (pipe)

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

## Library Functions

**Python:** fork() and wait() as in C programs

fork() starts 2nd Python process und executes the same Python script in it

```
#!/usr/bin/python
import os
import time
pid=os.fork()
if pid==0:
  time.sleep(5)
  print ">> I‘m the child."
else:
  print "I‘m the father. My child has PID ",pid
  print "Now I wait for the child..."
  os.wait()
  print "It finished."
```