

```

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 16:17:11 amd64 sshd[10140]: Accepted rsa for esser from :ffff:87.234.201.207 port 63375
Sep 20 16:17:11 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:18:10 amd64 sshd[10140]: Accepted rsa for esser from :ffff:87.234.201.207 port 63546
Sep 21 01:00:01 amd64 /usr/sbin/cron[17878]: (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[13269]: 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[13269]: Accepted rsa for esser from :ffff:87.234.201.207 port 64391
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[4674]: (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 18:04:05 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[124616]: (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 kernel: snd_seq_midi_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 kernel: snd_seq_oss: unsupported module, tainting kernel.
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[462]: (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

```

```

Sep 19 14:20:19 amd64 sshd[20944]: Accepted rsa for esser from :ffff:87.234.201.207 port 61507
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 16:17:11 amd64 sshd[10140]: Accepted rsa for esser from :ffff:87.234.201.207 port 63375
Sep 20 16:17:11 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:18:10 amd64 sshd[10140]: Accepted rsa for esser from :ffff:87.234.201.207 port 63546
Sep 21 01:00:01 amd64 /usr/sbin/cron[17878]: (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[13269]: 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[13269]: Accepted rsa for esser from :ffff:87.234.201.207 port 64391
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[4674]: (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 18:04:05 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[124616]: (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 kernel: snd_seq_midi_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 kernel: snd_seq_oss: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 kernel: snd_seq_oss: unsupported module, tainting kernel.
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[462]: (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

```

# 2. Prozesse und Threads

## Prozesse & Threads: Gliederung

**Vorlesung heute** – Einleitung  
– Theorie bzw. Grundlagen

**Praktikum 27./28.03.** – Prozesse auf der Linux-Shell  
– Prozesse in C-Programmen

**Vorlesung heute** – Prozesse & Threads  
unter Linux

**Praktikum 03./04.04.** – Threads in C-Programmen

Hans-Georg Eber, Hochschule München      Betriebssysteme I, SS 2008      2. Prozesse (2/2) – Folie 2

## Prozesse und Threads erzeugen (1/12)

- Neuer Prozess: `fork ()`

```

main()
{
    int pid = fork(); /* Sohnprozess erzeugen */
    if (pid == 0)
    {
        printf("Ich bin der Sohn, meine PID ist %d.\n", getpid() );
    }
    else
    {
        printf("Ich bin der Vater, mein Sohn hat die PID %d.\n",
            pid);
    }
}

```

Hans-Georg Eber, Hochschule München      Betriebssysteme I, SS 2008      2. Prozesse (2/2) – Folie 4

## Prozesse und Threads erzeugen (2/12)

- Anderes Programm starten: `fork` + `exec`

```
main() {
    int pid=fork(); /* Sohnprozess erzeugen */
    if (pid == 0) {
        /* Sohn startet externes Programm */
        exec1( "/usr/bin/gedit", "/etc/fstab", (char *) 0 );
    }
    else {
        printf("Es sollte jetzt ein Editor starten...\n");
    }
}
```

- Andere Betriebssysteme oft nur: `spawn`

```
main() {
    int ecode=spawn("c:\system\test.exe"); /* Sohn erzeugen */
}
```

## Prozesse und Threads erzeugen (4/12)

Wirklich mehrere Prozesse:

Nach `fork` () zwei Prozesse in der Prozessliste

```
> ps tree | grep simple
    |           |           | -bash---simplefork---simplefork
```

```
> ps w | grep simple
25684 pts/16 S+      0:00 ./simplefork
25685 pts/16 S+      0:00 ./simplefork
```

## Prozesse und Threads erzeugen (3/12)

Warten auf Sohn-Prozess: `wait` ()

```
#include <unistd.h>          /* sleep() */
main()
{
    int pid=fork();          /* Sohnprozess erzeugen */
    if (pid == 0)
    {
        sleep(2);            /* 2 sek. schlafen legen */
        printf("Ich bin der Sohn, meine PID ist %d\n", getpid() );
    }
    else
    {
        printf("Ich bin der Vater, mein Sohn hat die PID %d\n", pid);
        wait();              /* auf Sohn warten */
    }
}
```

## Prozesse und Threads erzeugen (5/12)

Linux: `pthread`-Bibliothek (POSIX Threads)

	Thread	Prozess
Erzeugen	<code>pthread_create()</code>	<code>fork()</code>
Auf Ende warten	<code>pthread_join()</code>	<code>wait()</code>

- Bibliothek einbinden:

```
#include <pthread.h>
```

- Kompilieren:

```
gcc -lpthread -o prog prog.c
```

## Prozesse und Threads erzeugen (6/12)

- Neuer Thread:  
`pthread_create()` erhält als Argument eine Funktion, die im neuen Thread läuft.
- Auf Thread-Ende warten:  
`pthread_join()` wartet auf einen *bestimmten* Thread.

## Prozesse und Threads erzeugen (8/12)

```
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

void *thread_function1(void *arg) {
    int i;
    for ( i=0; i<10; i++ ) {
        printf("Thread 1 sagt Hi!\n");
        sleep(1);
    }
    return NULL;
}

void *thread_function2(void *arg) {
    int i;
    for ( i=0; i<10; i++ ) {
        printf("Thread 2 sagt Hallo!\n");
        sleep(1);
    }
    return NULL;
}

int main(void) {
    pthread_t mythread1;
    pthread_t mythread2;

    if ( pthread_create( &mythread1, NULL,
        thread_function1, NULL) ) {
        printf("Fehler bei Thread-Erzeugung.");
        abort();
    }

    if ( pthread_create( &mythread2, NULL,
        thread_function2, NULL) ) {
        printf("Fehler bei Thread-Erzeugung.");
        abort();
    }

    pthread_join( mythread1, NULL );
    pthread_join( mythread2, NULL );

    printf("Thread 1 ist weg\n");
    printf("Thread 2 ist weg\n");

    exit(0);
}
```

## Prozesse und Threads erzeugen (7/12)

1. Thread-Funktion definieren:

```
void *thread_funktion(void *arg) {
    ...
    return ...;
}
```

2. Thread erzeugen:

```
pthread_t thread;

if ( pthread_create( &thread, NULL,
    thread_funktion, NULL) ) {
    printf("Fehler bei Thread-Erzeugung.\n");
    abort();
}
```

## Prozesse und Threads erzeugen (9/12)

### Keine „Vater-“ oder „Kind-Threads“

- POSIX-Threads kennen keine Verwandtschaft wie Prozesse (Vater- und Sohnprozess)
- Zum Warten auf einen Thread ist Thread-Variable nötig: `pthread_join (thread, ..)`

## Prozesse und Threads erzeugen (10/12)

Prozess mit mehreren Threads:

- Nur ein Eintrag in normaler Prozessliste
- Status: „l“, multi-threaded
- Über `ps -eLf` Thread-Informationen
  - NLWP: Number of light weight processes
  - LWP: Thread ID

```
> ps auxw | grep thread
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
esser    12022  0.0  0.0  17976   436 pts/15    Sl+  22:58   0:00 ./thread

> ps -eLf | grep thread
UID      PID  PPID  LWP  C  NLWP  STIME TTY          TIME CMD
esser   12166  4031 12166  0   3  23:01 pts/15    00:00:00 ./thread1
esser   12166  4031 12167  0   3  23:01 pts/15    00:00:00 ./thread1
esser   12166  4031 12177  0   3  23:01 pts/15    00:00:00 ./thread1
```

## Prozesse und Threads erzeugen (12/12)

Posix-Thread vs. Kernel-Thread:

- Ein mit `clone` erzeugter (Kernel-) Thread ist nicht dasselbe wie ein mit `pthread_create` erzeugter Posix-Thread!
- Posix-Bibliothek muss das gewünschte (Standard-) Verhalten über die von Linux bereitgestellten (`clone`-/Kernel-) Threads implementieren.

## Prozesse und Threads erzeugen (11/12)

Unterschiedliche Semantik:

- Prozess erzeugen mit `fork ()`
  - erzeugt zwei (fast) identische Prozesse,
  - beide Prozesse setzen Ausführung an gleicher Stelle fort (nach Rückkehr aus `fork`-Aufruf)
- Thread erzeugen mit `pthread_create (... , funktion, ...)`
  - erzeugt neuen Thread, der in die angeg. Funktion springt
  - erzeugender Prozess setzt Ausführung hinter `pthread_create`-Aufruf fort

## Prozessliste (1/8)

Kernel unterscheidet nicht zwischen Prozessen und Threads.

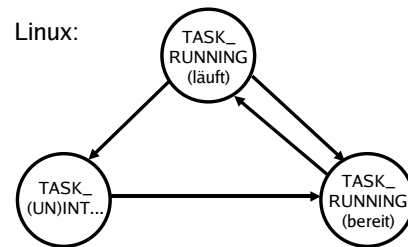
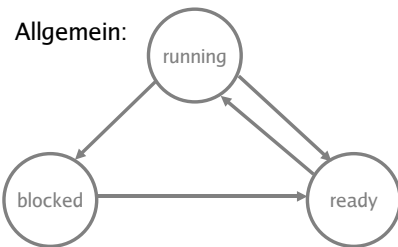
- Doppelt verkettete, ringförmige Liste
- Jeder Eintrag vom Typ `struct task_struct`
- Typ definiert in `include/linux/sched.h`
- Enthält alle Informationen, die Kernel benötigt
- `task_struct`-Definition 132 Zeilen lang!
- Maximale PID: 32767 (short int)

## Prozessliste (2/8)

Auszug aus `include/linux/sched.h`:

```
#define TASK_RUNNING      0
#define TASK_INTERRUPTIBLE 1
#define TASK_UNINTERRUPTIBLE 2
#define TASK_STOPPED     4
#define TASK_TRACED      8
/* in tsk->exit_state */
#define EXIT_ZOMBIE      16
#define EXIT_DEAD       32
/* in tsk->state again */
#define TASK_NONINTERACTIVE 64
#define TASK_DEAD       128
```

- `TASK_RUNNING`: ready oder running
- `TASK_INTERRUPTIBLE`: entspricht blocked
- `TASK_UNINTERRUPTIBLE`: auch blocked
- `TASK_STOPPED`: angehalten (z. B. von einem Debugger)
- `TASK_ZOMBIE`: beendet, aber Vater hat Rückgabewert nicht gelesen



## Prozessliste (4/8)

### Verwandtschaftsverhältnisse (neue Linux-Version)

```
struct task_struct {
    [...]
    struct task_struct *parent; /* parent process */
    struct list_head children; /* list of my children */
    struct list_head sibling; /* linkage in my parent's children list */
}
```

#### Zugriff auf alle Kinder:

```
list_for_each(list, &current->children) {
    task = list_entry(list, struct task_struct, sibling);
    /* task zeigt jetzt auf eines der Kinder */
}
```

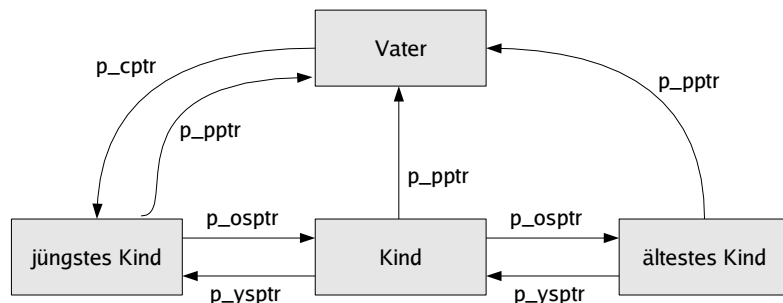
#### Vom aktuellen Pfad durch den Prozessbaum bis zu init:

```
for (task = current; task != &init_task; task = task->parent) {
    ...
}
```

## Prozessliste (3/8)

### Verwandtschaftsverhältnisse (alte Linux-Version)

```
struct task_struct {
    [...]
    struct task_struct *p_optr, *p_pptr, *p_cptr, *p_ysptr, *p_osptr;
```



## Prozessliste (5/8)

### Prozessgruppen und Sessions

```
struct task_struct {
    [...]
    struct task_struct *group_leader; /* threadgroup leader */
    [...]
    /* signal handlers */
    struct signal_struct *signal;
}

struct signal_struct {
    /* job control IDs */
    pid_t pgrp; /* Process Group ID */
    pid_t tty_old_pgrp;
    pid_t session; /* Session ID */
    /* boolean value for session group leader */
    int leader;
}
```

- Jeder Prozess Mitglied einer Prozessgruppe
- Process Group ID (PGID) – `ps j`
- `current->signal->pgrp`

## Prozessliste (6/8)

### Prozessgruppen

- Signale an alle Mitglieder einer Prozessgruppe:  
`killpg(pgrp, sig);`
- Warten auf Kinder aus der eigenen Prozessgruppe:  
`waitpid(0, &status, ...);`
- oder einer speziellen Prozessgruppe:  
`waitpid(-pgrp, &status, ...);`

## Prozessliste (8/8)

```
> ps j
PPID  PID  PGID  SID  TTY      TPGID STAT  UID   TIME COMMAND
19287  7628  7628  19287 pts/8    19287 S      500   0:00 /bin/sh /usr/bin/mozilla -mail
   7628  7637  7628  19287 pts/8    19287 Sl     500   20:50 /opt/moz/lib/mozilla-bin -mail
  9634 10095 10095 10095 tty1     10114 Ss     500   0:00 -bash
10095 10114 10114 10095 tty1     10114 S+     500   0:00 /bin/sh /usr/X11R6/bin/startx
10095 10115 10114 10095 tty1     10114 S+     500   0:00 tee /home/esser/.X.err
10114 10135 10114 10095 tty1     10114 S+     500   0:00 xinit /home/esser/.xinitrc
10135 10151 10151 10095 tty1     10114 S      500   0:00 /bin/sh /usr/X11R6/bin/kde
10151 10238 10151 10095 tty1     10114 S      500   0:00 kwrapper ksmsserver
10258 10270 10270 10270 pts/2    10270 Ss+    500   0:00 bash
10276 10278 10278 10278 pts/4    10278 Ss+    500   0:00 bash
10260 10284 10284 10284 pts/5    10284 Ss+    500   0:00 bash
10275 10292 10292 10292 pts/6    10989 Ss     500   0:00 bash
10259 10263 10263 10263 pts/1    10263 Ss+    500   0:00 bash
10263 28869 28869 10263 pts/1    10263 S      500   0:16 konqueror /media/usbdisk/dcim
10263 28872 28872 10263 pts/1    10263 S      500   0:13 konqueror /home/esser
29201 29203 29203 29203 pts/7    29203 Ss+    500   0:00 bash
  4822  4823  4823  4823 pts/14   4823 Ss+    500   0:00 -bash
  4823 31118 31118 4823  pts/14   4823 S      500   0:00 nedit kernel/sched.c
  4823 31297 31297 4823  pts/14   4823 S      500   0:00 nedit kernel/fork.c
23115 32703 32703 23115 pts/13   32703 R+     500   0:00 ps j
```

## Prozessliste (7/8)

### Sessions

- Meist beim Starten einer Login-Shell neu erzeugt
- Alle Prozesse, die aus dieser Shell gestartet werden, gehören zur Session
- Gemeinsames „kontrollierendes TTY“

## Prozesserzeugung (1/2)

Wichtigste Datei in den Kernel-Quellen: `kernel/fork.c`  
(enthält u. a. `copy_process`)

- `fork()` ruft `clone()` auf,
- `clone()` ruft `do_fork()` auf, und
- `do_fork()` ruft `copy_process()` auf

## Prozesserzeugung (2/2)

`copy_process()` macht:

- `dup_task_struct()`: neuer Kernel Stack, `thread_info` Struktur, `task_struct`-Eintrag
- Kind-Status auf `TASK_UNINTERRUPTIBLE`
- `copy_flags()`: `PF_FORKNOEXEC`
- `get_pid()`: Neue PID für Kind vergeben
- Je nach `clone()`-Parametern offene Dateien, Signal-Handler, Prozess-Speicherbereiche etc. kopieren oder gemeinsam nutzen
- Verbleibende Rechenzeit aufteilen (→ Scheduler)

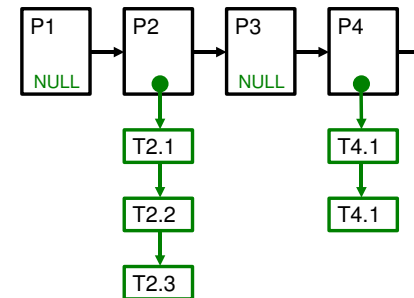
Danach: aufwecken, starten (Kind kommt vor Vater dran)

## Threads im Kernel (2/3)

- Fundamental anders als z. B. Windows und Solaris

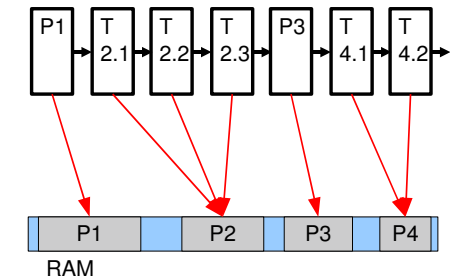
Modell 1:

reine Prozesslisten



Modell 2 (Linux):

Prozesse + Threads gemischt



## Threads im Kernel (1/3)

- Linux verwendet für Threads und Prozesse die gleichen Verwaltungsstrukturen (task list)
- Thread: Prozess, der sich mit anderen Prozessen bestimmte Ressourcen teilt, z. B.
  - virtueller Speicher
  - offene Dateien
- Jeder Thread hat `task_struct` und sieht für den Kernel wie ein normaler Prozess aus

## Threads im Kernel (3/3)

- Thread-Erzeugung: auch über `clone()`
- einfach andere Aufrufparameter:
  - Prozess: `fork` ->  
`clone(SIGCHLD, 0);`
  - Thread:  
`clone(CLONE_VM | CLONE_FS | CLONE_FILES | CLONE_SIGHAND, 0);`  
(`vm`: virtual memory, `fs`: Dinge wie Arbeitsverzeichnis, `Umask`, `Root-Verzeichnis` des Prozesses, `files`: offene Dateien, `sighand`: Signal Handlers)