

```
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[1014]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63546
Sep 21 01:00:01 amd64 /usr/sbin/cron[29278]: (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[30103]: (root) CMD (/sbin/evlogmgr -c 'age > "30d"')
Sep 21 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 02:00:01 amd64 /usr/sbin/cron[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 17:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 18:43:26 amd64 sshd[31263]: 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 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 "severity=DEBUG")
Sep 22 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 20:23:23 amd64 /usr/sbin/cron[12436]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 23 01:00:01 amd64 /usr/sbin/cron[12436]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 23 01:00:01 amd64 /usr/sbin/cron[12436]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 23 01:00:01 amd64 /usr/sbin/cron[12436]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 23 02:00:01 amd64 /usr/sbin/cron[12436]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 23 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:05 amd64 sshd[6554]: Accepted pubkey 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[12436]: (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[862]: (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[11606]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:33 amd64 sshd[11670]: 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
```

Projekt: Grundlagen

Übersicht

- Datenstrukturen für Threads
- Context Switch und Stacks
- Timer (alarm, setitimer) unter Linux
- Signal Handler

9. Projekt: Thread-Bibliothek

Programmier-Projekt

- Implementierung einer User-Level-Thread-Bibliothek (ähnlich POSIX Threads)
 - Threads erzeugen mit `pthread_create()`
 - Threads beenden mit `pthread_exit()`
 - auf Thread warten mit `pthread_join()`
 - Thread-ID abfragen mit `pthread_self()`
- Zur Implementation gehört:
 - Round-Robin-Scheduler für Threads
 - Timer (für unterbrechenden Scheduler)

Datenstrukturen

- Für jeden Thread einen Thread Control Block (TCB) verwalten
 - Zustandsinformationen zum Thread (blockiert, bereit etc.)
 - Thread-Context
 - Register
 - Stack, Stack Pointer, Base Pointer
 - Verwaltungsdaten (z. B. Priorität)
- `pthread_create` muss einen neuen TCB erzeugen

Stack (1)

- Programme nutzen den Stack über zwei Register:
 - **esp**, zeigt immer auf aktuelles Ende des Stacks (Stack „wächst“ nach unten)
 - **ebp**, zeigt immer auf aktuellen Stack Frame; erlaubt einer Funktion schnellen Zugriff auf
 - Aufruf-Argumente (oberhalb ebp) und
 - lokale Variablen (unterhalb ebp)

Stack (3)

```
// ebp-test.c
#include <stdio.h> // printf

#define get_ebp(v) \
__asm__ ("mov %%ebp, %0" : "=r"(v))

int f (int x, int y) {
    unsigned a, b;

    get_ebp (a);
    printf ("%a : %8p \n",    &a);
    printf ("%b : %8p \n\n",  &b);
    printf ("%ebp : 0x%08x \n\n", a);
    printf ("%x : %8p \n",    &x);
    printf ("%y : %8p \n",    &y);
    return 0;
}

int main () {
    f (12, 24);
}
```

```
$ ./ebp-test
&a : 0xbfad6248
&b : 0xbfad624c
ebp : 0xbfad6258
&x : 0xbfad6260
&y : 0xbfad6264
```

Stack (2)

- Bei Funktionsaufrufen wird der Stack benutzt:
 - Aufruf `func (a, b, c)` schreibt auf den Stack:
 - c, b, a (umgekehrte Reihenfolge)
 - Rücksprungadresse (Befehl nach `func(a, b, c)`)
 - `func` sichert alten **ebp** auf Stack
 - `func` reserviert Platz für lokale Variablen auf Stack, und zwar durch Anpassen von **esp**
 - in `func` Zugriff auf
 - Argumente: z. B. `+4(ebp)`, `+8(ebp)`, ...
 - lokale Variablen: z. B. `-4(ebp)`, `-8(ebp)`, ...

Stack- und Base-Pointer

```
int f (int x) {
    int y = 2;
    return x+y;
}

int main () {
    int a = f (42);
    return a;
}

leave =
movl   %ebp, %esp
popl   %ebp

f:
pushl  %ebp           // ebp sichern
movl   %esp, %ebp    // ebp = esp
subl   $16, %esp     // esp -= 16
movl   $2, -4(%ebp)  // y = 2
movl   -4(%ebp), %eax // eax = y
movl   8(%ebp), %edx  // edx = x
addl   %edx, %eax    // eax += edx
leave
ret

main:
pushl  %ebp
movl   %esp, %ebp
subl   $20, %esp
movl   $42, (%esp)   // 42 auf Stack
call   f             // f(42) in eax
movl   %eax, -4(%ebp) // a = eax
movl   -4(%ebp), %eax // eax = a
leave
ret
```

Threads und Stacks

- Jeder Thread braucht seinen eigenen Stack (sonst: Verwirrung beim Umschalten auf einen anderen Thread)
- Der „erste“ Stack kommt automatisch; für jeden neuen Thread zusätzlichen Stack-Speicher allozieren (`malloc`)
- Beim Wechsel von einem Thread zum anderen auch zum richtigen Stack umschalten
- In der Übung: Stacks bei `pthread`-Implem.

Signal-Handler

- Über `alarm()` kann jeder Prozess Signal-Handler für verschiedene Signale eintragen
- Uns interessiert das Signal `SIGALRM` (14)
- Eintragen des Handlers mit:

```
#include <signal.h> // signal (), SIGALRM
signal (SIGALRM, alarm_handler)
```
- Handler-Funktion implementieren:

```
void alarm_handler (int sig) {
    ...
}
```

Timer und Signal-Handler

- Umschalten zwischen mehreren Threads über Timer
- ähnlich wie bei Prozess-Scheduler: Der wird über Timer-Interrupts realisiert
- für User-Level-Threads geht es nicht mit Timer-Interrupts (Kernel kennt die Threads nicht), Alternative: Alarm-Signal und Timer:
 - Signal-Handler für das Signal `SIGALRM`
 - Regelmäßig dieses Signal erzeugen

Timer erzeugen: `alarm()`

- Der Aufruf `alarm(sec)` erzeugt einen (einmaligen) Timer, der nach `sec` Sekunden auslöst
- Damit der Timer regelmäßig auftritt, muss der Signal-Handler den Alarm erneuern
- `alarm()` erlaubt nur sekundengenaue Angaben, besser für Scheduling ist die Funktion `setitimer()`

Timer erzeugen: setitimer()

- `setitimer()` erwartet (u. a.) ein Argument vom Typ `struct itimerval`:

```
struct itimerval {
    struct timeval it_interval; /* next value */
    struct timeval it_value;   /* current value */
};

struct timeval {
    long tv_sec;               /* seconds */
    long tv_usec;             /* microseconds */
};
```

- Das erlaubt mikrosekunden-genaue Aufrufe

Thread-Scheduler

- Idee: `setitimer()` erzeugt regelmäßig ein `SIGALRM`-Signal
- Handler-Funktion für dieses Signal ist unser Scheduler
- Scheduler wählt neuen Thread aus und sorgt dafür, dass Prozess (nach Verlassen des Handlers) nicht zu aktuellem Thread, sondern zu neuem zurück kehrt
- Problem: Wie kriegt man das hin? → Genaues Verständnis des Stack-Aufbaus nötig

setitimer-Beispiel

- Timer zählt Echtzeit (`ITIMER_REAL`) oder nur die Zeit, in der der Prozess läuft (`ITIMER_VIRTUAL`) – für Scheduler ist die Prozesszeit interessanter
- Timer nicht nur einmalig, sondern mit Intervall

```
#include <sys/time.h>

struct itimerval timer;
timer.it_value.tv_sec = 0;
timer.it_value.tv_usec = 250000; // Alarm nach 250 msec
timer.it_interval.tv_sec = 0;
timer.it_interval.tv_usec = 100000; // dann alle 100 msec
setitimer (ITIMER_VIRTUAL, &timer, NULL);
```