

```

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[6499]: 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 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[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 64333
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[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[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 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[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

```

# 8. Threads

## Threads (1)

### Was ist ein Thread?

- Aktivitätsstrang in einem Prozess
- einer von mehreren
- Gemeinsamer Zugriff auf Daten des Prozess
- aber: Stack, Befehlszähler, Stack Pointer, Hardware-Register separat pro Thread
- Prozess-Scheduler verwaltet Threads – oder nicht (Kernel- oder User-level-Threads)

## Threads

## Threads (2)

- Grundlagen, POSIX-Threads
- pthread\_create(), pthread\_join()
- Synchronisation mit Mutex, Semaphor

### Warum Threads?

- Multi-Prozessor-System: Mehrere Threads echt gleichzeitig aktiv
- Ist ein Thread durch I/O blockiert, arbeiten die anderen weiter
- Besteht Programm logisch aus parallelen Abläufen, ist die Programmierung mit Threads einfacher

## Threads (3): Beispiele

### Zwei unterschiedliche Aktivitätsstränge: Komplexe Berechnung mit Benutzeranfragen

Ohne Threads:

```
while (1) {
    rechne_ein_bisschen ();
    if benutzereingabe (x) {
        bearbeite_eingabe (x)
    }
}
```

## Threads (5): Beispiele

### Server-Prozess, der viele Anfragen bearbeitet

- Prozess öffnet Port
- Für jede eingehende Verbindung: Neuen Thread erzeugen, der diese Anfrage bearbeitet
- Nach Verbindungsabbruch Thread beenden
- Vorteil: Keine Prozess-Erzeugung (Betriebssystem!) nötig

## Threads (4): Beispiele

### Komplexe Berechnung mit Benutzeranfragen

Mit Threads:

<pre>T1: while (1) {     rechne_alles (); }</pre>	<pre>T2: while(1) {     if benutzereingabe (x) {         bearbeite_eingabe (x);     } }</pre>
---	---

## Threads (6): Beispiel MySQL

Ein Prozess, neun Threads:

```
[esser:~]$ ps -elf | grep mysql
UID      PID  PPID  LWP  C  NLWP STIME TTY          TIME CMD
root    27833    1 27833  0   1 Jan04 ?        00:00:00 /bin/sh /usr/bin/mysqld_safe
mysql   27870 27833 27870  0   9 Jan04 ?        00:00:00 /usr/sbin/mysqld --basedir=/usr
mysql   27870 27833 27872  0   9 Jan04 ?        00:00:00 /usr/sbin/mysqld --basedir=/usr
mysql   27870 27833 27873  0   9 Jan04 ?        00:00:00 /usr/sbin/mysqld --basedir=/usr
mysql   27870 27833 27874  0   9 Jan04 ?        00:00:00 /usr/sbin/mysqld --basedir=/usr
mysql   27870 27833 27875  0   9 Jan04 ?        00:00:00 /usr/sbin/mysqld --basedir=/usr
mysql   27870 27833 27876  0   9 Jan04 ?        00:00:00 /usr/sbin/mysqld --basedir=/usr
mysql   27870 27833 27877  0   9 Jan04 ?        00:00:00 /usr/sbin/mysqld --basedir=/usr
mysql   27870 27833 27878  0   9 Jan04 ?        00:00:00 /usr/sbin/mysqld --basedir=/usr
mysql   27870 27833 27879  0   9 Jan04 ?        00:00:00 /usr/sbin/mysqld --basedir=/usr
[esser:~]$
```

PID: Process ID  
PPID: Parent Process ID  
LWP: Light Weight Process ID (Thread-ID)  
NLWP: Number of Light Weight Processes

## Unterschied Prozesse / Threads (1/2)

- Parallel programmieren wahlweise mit mehreren Prozessen / mehreren Threads
- Austausch / Kommunikation untereinander
  - **Prozesse:** kein gemeinsamer Speicher. Austausch z. B. über Nachrichten, Zugriff auf Datei
  - **Threads:** gemeinsamer Speicher, Austausch z. B. durch direktes Auslesen von Variablen

## POSIX-Threads

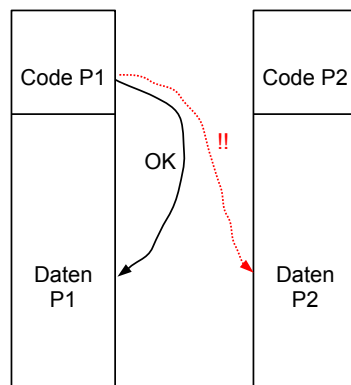
Linux: pthread-Bibliothek (POSIX Threads)

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

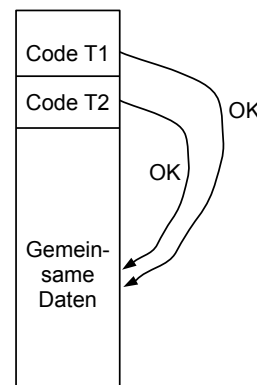
- Bibliothek einbinden:  
`#include <pthread.h>`
- Kompilieren:  
`gcc -o prog prog.c -lpthread`

## Unterschied Prozesse / Threads (2/2)

Zwei Prozesse



Zwei Threads



## POSIX-Threads

- 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.

# POSIX-Threads

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();  
}
```

```
#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();  
    }  
}
```

```
sleep(5);  
  
if ( pthread_create( &mythread2, NULL,  
    thread_function2, NULL) ) {  
    printf("Fehler bei Thread-Erzeugung.");  
    abort();  
}  
  
sleep(5);  
printf("bin noch hier...\n");  
  
if ( pthread_join ( mythread1, NULL ) ) {  
    printf("Fehler beim Join.");  
    abort();  
}  
  
printf("Thread 1 ist weg\n");  
  
if ( pthread_join ( mythread2, NULL ) ) {  
    printf("Fehler beim Join.");  
    abort();  
}  
  
printf("Thread 2 ist weg\n");  
  
exit(0);  
}
```

# POSIX-Threads

## 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, ...)`

# POSIX-Threads

## 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    S1+  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
```

# POSIX-Threads

## 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

# POSIX-Semaphore

- Deklarieren: `sem_t s;`
- Initialisieren: `sem_init (&s, 0, Start);`
- Erniedrigen oder blockieren (falls =0):  
`sem_wait (&s);`
- Erhöhen oder wartenden Thread wecken (falls Threads in Warteschlange):  
`sem_post (&s);`

# Synchronisation

- Zugriff auf gemeinsame Ressourcen (z. B. prozessweit gültige Variablen in mehreren Threads) einschränken
- POSIX-Semaphore (auch prozess-übergreifend nutzbar)
- POSIX-Thread-Mutexe (nur innerhalb eines Prozesses nutzbar)
- Beispiel für Synchronisationsproblem

# POSIX-Semaphor als Mutex

## Mutex: mutual exclusion

- Deklarieren: `sem_t m;`
- Initialisieren: `sem_init (&m, 0, 1);`
- Erniedrigen oder blockieren (falls =0):  
`sem_wait (&m);`
- Erhöhen oder wartenden Thread wecken (falls Threads in Warteschlange):  
`sem_post (&m);`

# Oder: POSIX-Thread-Mutex

## Eigener Mutex-Typ:

- Deklarieren: `pthread_mutex_t m;`
- Initialisieren: `pthread_mutex_init (&m, NULL);`
- Sperren: `pthread_mutex_lock (&m);`
- Freigeben: `pthread_mutex_unlock (&m);`

# Lösung mit POSIX-Semaphor

```
sem_t mutex;

void* addierer () {
    int i, tmp;
    for (i = 0; i < ITERATIONEN; i++) {
        sem_wait (&mutex); // Zugriff sperren
        tmp = count;
        tmp++;
        count = tmp;
        sem_post (&mutex); // Zugriff wieder freigeben
    }
}

int main () {
    sem_init (&mutex, 0, 1); // Semaphor einrichten
    pthread_t t1, t2;
    pthread_create(&t1, NULL, addierer, NULL);
    pthread_create(&t2, NULL, addierer, NULL);
    [...]
```

```
root@ubu64:/home/esser/thread-sem# ./a.out
OK: count = 20000000
```

# Beispiel: Synchr.-Problem

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#define ITERATIONEN 10000000

int count = 0;
sem_t mutex;

void* addierer () {
    int i, tmp;
    for (i = 0; i < ITERATIONEN; i++) {
        tmp = count;
        tmp++;
        count = tmp;
    }
}

int main () {
    pthread_t t1, t2;
    pthread_create (&t1, NULL, addierer, NULL);
    pthread_create (&t2, NULL, addierer, NULL);
    pthread_join (t1, NULL);
    pthread_join (t2, NULL);

    if (count != 2 * ITERATIONEN)
        printf ("Fehler: count = %d, "
                "sollte aber %d sein.\n",
                count, 2*ITERATIONEN);
    else
        printf ("OK: count = %d\n", count);
    return 0;
}
```

```
root@ubu64:/home/esser/thread-sem# ./a.out
Fehler: count = 11499212, sollte aber 20000000 sein.
root@ubu64:/home/esser/thread-sem# ./a.out
Fehler: count = 11841573, sollte aber 20000000 sein.
```

# Lösung mit POSIX-Thread-Mutex

```
pthread_mutex_t mutex;

void* addierer () {
    int i, tmp;
    for (i = 0; i < ITERATIONEN; i++) {
        pthread_mutex_lock (&mutex); // Zugriff sperren
        tmp = count;
        tmp++;
        count = tmp;
        pthread_mutex_unlock (&mutex); // Zugriff wieder freigeben
    }
}

int main () {
    pthread_mutex_init (&mutex, NULL); // Mutex einrichten
    pthread_t t1, t2;
    pthread_create(&t1, NULL, addierer, NULL);
    pthread_create(&t2, NULL, addierer, NULL);
    [...]
```

```
root@ubu64:/home/esser/thread-sem# ./a.out
OK: count = 20000000
```