

PR: Programmiersprache C (Linux)  
Einheit 10: Threads

Roland A. Eggetsberger  
Inst. f. Informationsverarbeitung und Mikroprozessortechnik (FIM)  
Johannes Kepler Univ. Linz, Altenbergerstr. 69, A-4040 Linz, Austria

### Threads

- Prozesse und Threads
- Thread Levels
- Threads
- Threadspezifische Daten
- Identifiers
- Priority
- Signals
- Terminierung

---



---



---



---

### Prozesse und Threads (I)

- Prozesse
  - Eigener Adressraum
  - Kontrolle der Ressourcen
  - Kommunikation mit anderen Prozessen

### Prozesse und Threads (II)

- Threads
  - Gemeinsamer Adressraum
  - Zugriff auf Ressourcen
  - Kommunikation effizienter

---



---



---



---

### Thread Levels (I)

- User-Level Threads
  - Thread Verwaltung durch die Applikation
  - Scheduling von Applikation abhängig
  - Braucht nur eine entsprechende Library

### Thread Levels (II)

- Kernel-Level Threads
  - Verwaltung durch den Kernel
  - Kernel Scheduling

---



---



---



---



---



---

## Threads (I)

- Thread erzeugen (I)

```
#include <pthread.h>
int pthread_create(pthread_t *thread,
pthread_attr_t *attr, void *
(*start_routine)(void *), void *arg);
```

Thread mit gewissen Attributen starten

## Threads (II)

- Thread erzeugen (II)

Dabei wird die Funktion `start_routine` mit erstem Argument `arg` aufgerufen  
`thread` liefert eine ID zur weiteren Verwendung  
`NULL` als Attribut liefert einen Default Thread

## Threads (III)

- Thread beenden

```
#include <pthread.h>
void pthread_exit(void *retval);
```

Explizites beenden eines Threads mit Rückgabewert `retval`  
 Im Defaultfall wird der Thread mit der Startroutine beendet

## Threads (IV)

- Warten

```
#include <pthread.h>
int pthread_join(pthread_t th, void
**thread_return);
```

Warten bis Thread `th` terminiert  
 Falls `thread_return` nicht `NULL` ist, wird der Rückgabewert von `th` referenziert

## Threads (V)

- Speicherfreigabe

```
#include <pthread.h>
int pthread_detach(pthread_t th);
```

Speicher vom Thread `th` kann nach dessen Terminierung verwendet werden  
 Rückgabewert 0 bzw. im Fehlerfall ungleich 0

## Threads (VI)

- Initialisierung

```
#include <pthread.h>
pthread_once_t once_control =
PTHREAD_ONCE_INIT;
int pthread_once(pthread_once_t
*once_control, void
(*init_routine)(void));
```

Ein Codestück wird nur einmal ausgeführt

## Threadspezifische Daten (I)

- Verwendung  
Threads besitzen einen gemeinsamen Datenbereich  
Oft werden aber auch private Daten benötigt

## Threadspezifische Daten (II)

- Schlüssel erzeugen

```
#include <pthread.h>
int pthread_key_create(pthread_key_t *key,
void (*destr_function) (void *));
```

Erzeugt einen neuen Schlüssel (i.e. void-Pointer)  
Eine Destruktor-Funktion zur Freigabe des Datenbereiches kann übergeben werden

## Threadspezifische Daten (III)

- Schlüssel löschen

```
#include <pthread.h>
int pthread_key_delete(pthread_key_t key);
```

Der Schlüssel wird gelöscht, für die Freigabe des Datenbereiches muss gesorgt werden

## Threadspezifische Daten (IV)

- Daten anbinden

```
#include <pthread.h>
int pthread_setspecific(pthread_key_t key,
const void *pointer);
```

Ein Datenbereich wird dem Schlüssel angebinden  
Bei mehrmaliger Verwendung ist der Speicher wieder freizugeben

## Threadspezifische Daten (V)

- Daten verwenden

```
#include <pthread.h>
void *pthread_getspecific(pthread_key_t
key);
```

Der Datenbereich kann mit dem erhaltenen Pointer angesprochen werden

## Threadspezifische Daten (VI)

- Beispiel (I)

```
/* Key for the thread-specific buffer */
static pthread_key_t buffer_key;

/* Once-only initialisation of the key */
static pthread_once_t buffer_key_once =
PTHREAD_ONCE_INIT;
```

### Threadspezifische Daten (VII)

- Beispiel (II)

```
/* Allocate the thread-specific buffer */
void buffer_alloc(void) {

    pthread_once(&buffer_key_once,
                buffer_key_alloc);
    pthread_setspecific(buffer_key,
                        malloc(100));
}
```

### Threadspezifische Daten (VIII)

- Beispiel (III)

```
/* Return the thread-specific buffer */
char *get_buffer(void){

    return (char *)
           pthread_getspecific(buffer_key);
}
```

### Threadspezifische Daten (IX)

- Beispiel (IV)

```
/* Allocate the key */
static void buffer_key_alloc() {

    pthread_key_create(&buffer_key,
                      buffer_destroy);
}
```

### Threadspezifische Daten (X)

- Beispiel (V)

```
/* Free the thread-specific buffer */
static void buffer_destroy(void *buf) {

    free(buf);
}
```

### Identifiers (I)

- Lesen

```
#include <pthread.h>
pthread_t pthread_self(void);
```

Gibt den Thread Identifier zurück

### Identifiers (II)

- Vergleichen

```
#include <pthread.h>
int pthread_equal(pthread_t thread1,
                  pthread_t thread2);
```

Vergleicht zwei Thread Identifier

## Priority (I)

- Yield

```
#include <sched.h>
int sched_yield(void);
```

Stoppen der Ausführung zugunsten eines Threads mit gleicher oder höherer Priorität

## Priority (II)

- Setzen (I)

```
#include <pthread.h>
int pthread_setschedparam(pthread_t
target_thread, int policy, const struct
sched_param *param);
```

Setzen der Priorität eines Threads

## Priority (III)

- Setzen (II)

Mögliche Policies

```
SCHED_OTHER ... regular, non-realtime
SCHED_RR ..... realtime, round-robin
SCHED_FIFO .... realtime, first-in first-
out
```

## Priority (IV)

- Abfragen

```
#include <pthread.h>
int pthread_getschedparam(pthread_t
target_thread, int *policy, struct
sched_param *param);
```

Abfragen der Priorität eines Threads

## Signals (I)

- Senden

```
#include <pthread.h>
#include <signal.h>
int pthread_kill(pthread_t thread,
int signo);
```

Senden an einen Thread

## Signals (II)

- Maskieren (I)

```
#include <pthread.h>
#include <signal.h>
int pthread_sigmask(int how, const
sigset_t *newmask, sigset_t *oldmask);
```

Maskierung in Abhängigkeit von how

## Signals (III)

- Maskieren (II)

Aktionen

```
SIG_SETMASK ... Setzen
SIG_BLOCK ..... Addieren
SIG_UNBLOCK ... Subtrahieren
```

## Terminierung (I)

- Senden

```
#include <pthread.h>
int pthread_cancel(pthread_t thread);
```

Terminierung an einen anderen Thread senden

## Terminierung (II)

- Terminierungsstatus (I)

```
#include <pthread.h>
int pthread_setcancelstate(int state,
                          int *oldstate);
```

Umgehen mit Terminierungsanfragen

## Terminierung (III)

- Terminierungsstatus (II)

Mögliche Zustände

```
PTHREAD_CANCEL_ENABLE .... Zulassen
PTHREAD_CANCEL_DISABLE ... Verweigern
```

## Terminierung (IV)

- Antworttyp (I)

```
#include <pthread.h>
int pthread_setcanceltype(int type,
                          int *oldtype);
```

Reagieren auf Terminierungsanfragen

## Terminierung (V)

- Antworttyp (II)

Mögliche Typen

```
PTHREAD_CANCEL_ASYNCHRONOUS ... Sofort
PTHREAD_CANCEL_DEFERRED ..... Warten
```

Terminierung (VI)

- Abfragen

```
#include <pthread.h>
void pthread_testcancel(void);
```

Anhängige Terminierung wird durchgeführt

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---