Systemnahe Programmierung in C

32 Nebenläufige Fäden – Praxis

J. Kleinöder, D. Lohmann, V. Sieh

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Sommersemester 2024

http://sys.cs.fau.de/lehre/ss24

Beispiel: POSIX Threads (pthread)

- Programmierschnittstelle standardisiert: **pthread-Bibliothek** (IEEE-POSIX-Standard P1003.4a)
- pthread-Schnittstelle (Basisfunktionen):

  `pthread_create:` Faden erzeugen
  `pthread_exit:` Faden beendet sich selbst
  `pthread_join:` auf Ende eines Fadens warten
  `...:`
- Funktionen in pthread-Bibliothek zusammengefasst

```
gcc ... -pthread ...
```


pthread-Schnittstelle

- Faden-Erzeugung

```
#include <pthread.h>

int pthread_create(pthread_t *tid, const pthread_attr_t *attr,
                   void *(*func)(void *), void *param);
```

- Parameter

  `tid`: Zeiger auf Variable, in der die Faden-ID abgelegt werden soll.

  `attr`: Zeiger auf Attribute (z.B. Stack-Größe) des Fadens. `NULL` für Standard-Attribute.

  `func`, `param`: Der neu erzeugte Faden führt die Funktion `func` mit dem Parameter `param` aus.

- Als Rückgabewert wird 0 geliefert. Im Fehlerfall wird ein Fehlercode (ähnlich `errno`) zurückgeliefert.

pthread-Schnittstelle

- Faden beenden (bei `return` aus `func` oder):

```
#include <pthread.h>

void pthread_exit(void *retval);
```

Der Faden wird beendet und `retval` wird als Rückgabewert zurückgeliefert (siehe `pthread_join`.

- Auf Faden warten und `pthread_exit`-Status abfragen:

```
#include <pthread.h>

int pthread_join(pthread_t tid, void **retvalp);
```

Wartet auf den Faden mit der Faden-ID `tid` und liefert dessen Rückgabewert über `retvalp` zurück.
Als Rückgabewert wird 0 geliefert. Im Fehlerfall wird ein Fehlercode (ähnlich `errno`) zurückgeliefert.

pthread-Beispiel

- Beispiel (Multiplikation Matrix mit Vektor; $\vec{c} = A\vec{b}$):

```
double a[100][100], b[100], c[100];

static void *mult(void *ci) {
    int i = (double *) ci - c;

    double sum = 0.0;
    for (int j = 0; j < 100; j++) {
        sum += a[i][j] * b[j];
    }
    c[i] = sum;
    return NULL;
}

int main(void) {
    pthread_t tid[100];

    for (int i = 0; i < 100; i++) {
        pthread_create(&tid[i], NULL, mult, &c[i]);
    }
    for (int i = 0; i < 100; i++) {
        pthread_join(tid[i], NULL);
    }
}
```

pthread-Koordinierung und -Synchronisierung

- Koordinierung durch Mutex-Variablen
  - Erzeugung von Mutex-Variablen

```
pthread_mutex_t m;
pthread_mutex_init(&m, NULL);
```

  - `lock`-Operation

```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *m);
```

  - `unlock`-Operation

```
#include <pthread.h>

int pthread_mutex_unlock(pthread_mutex_t *m);
```


pthread-Beispiel

- Mutex-Beispiel:

```
volatile int counter = 0;
pthread_mutex_t m;
pthread_mutex_init(&m, NULL);
```

```
...      /* Thread 1 */
pthread_mutex_lock(&m);
counter++;
pthread_mutex_unlock(&m);
...
```

```
...      /* Thread 2 */
pthread_mutex_lock(&m);
printf("counter = %d\n", counter);
counter = 0;
pthread_mutex_unlock(&m);
...
```


pthread-Koordinierung und -Synchronisierung (2)

- Synchronisierung durch Bedingungs-Variablen (Condition Variable)
  - auf eine Bedingung kann gewartet werden (sleep)
  - eine Bedingung kann signalisiert werden (wakeup)
  - Erzeugung einer Condition-Variablen

```
pthread_cond_t c;
pthread_cond_init(&c, NULL);
```

  - auf eine Bedingung warten

```
#include <pthread.h>

int pthread_cond_wait(pthread_cond_t *c, pthread_mutex_t *m);
```

  - eine Bedingung signalisieren

```
#include <pthread.h>

int pthread_cond_signal(pthread_cond_t *c);
int pthread_cond_broadcast(pthread_cond_t *c);
```

`pthread_cond_signal` weckt *einen* Faden, `pthread_cond_broadcast` weckt *alle* auf die Bedingung wartenden Fäden auf

pthread-Beispiel (2)

- Beispiel: zählende Semaphore

```
pthread_mutex_t m;
pthread_cond_t c;

pthread_mutex_init(&m, NULL);
pthread_cond_init(&c, NULL);
```

```
void P(volatile int *s) {
    pthread_mutex_lock(&m);
    while (*s == 0) {
        pthread_cond_wait(&c, &m);
    }
    *s -= 1;
    pthread_mutex_unlock(&m);
}
```

```
void V(volatile int *s) {
    pthread_mutex_lock(&m);
    *s += 1;
    pthread_cond_broadcast(&c);
    pthread_mutex_unlock(&m);
}
```

Fäden, Koordinierung, Synchronisierung in Java

- Faden-Konzept, Koordinierung und Synchronisierung in Java integriert
- Erzeugung von Fäden über die Thread-Klasse; Beispiel:

```
class MyClass implements Runnable {
    public void run() {
        System.out.println("Hello!");
    }
}
...
MyClass o = new MyClass();    // create object
Thread t1 = new Thread(o);    // create thread to run in o
t1.start();                   // start thread
Thread t2 = new Thread(o);    // create second thread
t2.start();                   // start second thread
```

Fäden, Koordinierung, Synchronisierung in Java (2)

- Koordinierung und Synchronisierung über jedes beliebige Objekt
  - Koordinierung über `synchronized`-Blöcke

```
synchronized(obj) {
    ...
}
```

    Ein solcher Block ruft zu Block-Beginn ein `lock` auf das Objekt `obj` auf, führt die angegebenen Anweisungen aus, und ruft vor dem Verlassen des Blockes das entsprechende `unlock` auf.
  - Synchronisierung über `wait`, `notify` und `notifyAll`

    `obj.wait()`: wartet auf die Signalisierung einer Bedingung auf dem angegebenen Objekt `obj`.

    `obj.notify()`: signalisiert eine Bedingung auf dem angegebenen Objekt `obj` an *einen* wartenden Faden.

    `obj.notifyAll()`: signalisiert eine Bedingung auf dem angegebenen Objekt `obj` *allen* wartenden Fäden.

Fäden, Koordinierung, Synchronisierung in Java (3)

- Beispiel Koordinierung und Synchronisierung:

```
public class Semaphore {
    private int s;

    public Semaphore(int s0) {
        s = s0;
    }
    public void P() {
        synchronized(this) {
            while (s == 0)
                this.wait();
            s--;
        }
    }
    public void V() {
        synchronized(this) {
            s++;
            this.notifyAll();
        }
    }
}
```

Entspricht dem pthread-Beispiel...

Fäden, Koordinierung, Synchronisierung in Java (4)

- Vereinfachte Schreibweise (entspricht „Monitor”-Konzept):

```
public class Semaphore {
    private int s;

    public Semaphore(int s0) {
        s = s0;
    }
    public synchronized void P() {
        while (s == 0) {
            wait();
        }
        s--;
    }
    public synchronized void V() {
        s++;
        notifyAll();
    }
}
```