

# Aufgabenblatt 6

## Echtzeitsysteme

Institut: Berliner Hochschule für Technik  
Dozent: Prof. Dr. Christian Forler  
Url: <https://lms.bht-berlin.de/>  
Email: [cforler@bht-berlin.de](mailto:cforler@bht-berlin.de)  
SoSe 2022

### Aufgabe 1 (2 Punkte) Mutex

Was passiert wenn die beiden Zeilen 13 (`pthread_mutex_lock(&mutex)`) und 17 (`pthread_mutex_unlock(&mutex)`) des folgenden Programms auskommentiert werden?

```
1 #include <pthread.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <unistd.h>
5
6 #define NTHRDS 5
7
8 int sharedData = 0;
9 pthread_mutex_t mutex;
10
11
12 void *add1000(void *n) {
13     pthread_mutex_lock(&mutex);
14     int j = sharedData;
15     sleep(rand() % 6);
16     sharedData = j + 1000;
17     pthread_mutex_unlock(&mutex);
18
19     puts("1000_added!");
20     return n;
21 }
22
23 int main() {
24     pthread_t thrd[NTHRDS];
25     int t;
26
27     pthread_mutex_init(&mutex, NULL);
28     srand(time(NULL));
29     for (t=0; t<NTHRDS; t++)
30         pthread_create(&thrd[t], NULL, add1000, NULL);
31     for (t=0; t<NTHRDS; t++) pthread_join(thrd[t], NULL);
32
33     printf("Shared_data_=%d\n", sharedData);
34     return 0;
35 }
```

### Aufgabe 2 (2 Punkte) Threadsichere Methoden

Schreiben Sie sich eine threadsichere Variante von `strtok()` und testen Sie diese ausgiebig.

### Aufgabe 3 (4 Punkte) Parallele Quersummenberechnung II

Erstellen Sie sich mit <https://www.random.org/integers/> eine Liste von 1000 Zufallszahlen zwischen 1 und 1000 und speichern sie diese unter `number.txt` ab. Schreiben Sie ein Programm welche die Datei `number.txt` als Kommandozeilenparameter einliest und mehrere Threads startet welche parallel die Quersummen der einzelnen Zahlen berechnen. Am Ende soll die Summe aller Quersummen ausgegeben werden.

### Aufgabe 4 (4 Punkte) Selbstjoin

Was passiert wenn ein Thread die Anweisung `pthread_join(pthread_self(), NULL)` ausführt? Überlegen Sie sich eine Antwort und verifizieren Sie diese mit Hilfe eines kleinen Programms.

### Aufgabe 5 (4 Punkte) Threadsichere Queue

Testen Sie, ob Ihre Queue-Implementierung aus Aufgabenblatt 1 threadsicher ist. Sollte dies nicht der Fall sein, dann machen Sie diese threadsicher.

### Aufgabe 6 (4 Punkte) Produzenten Konsumenten Problem (PKP)

Semaphore können auch mittels Bedingungsvariablen implementiert werden. Verwenden Sie die folgenden Semaphore-Implementation bestehend aus den Dateien (`semaphore.h` und `semaphore.c`) und schreiben sie ein Programm (`pkp`) welches das Produzenten Konsumenten Problem aus der Vorlesung löst. Implementieren Sie 2 Produzenten (`p1` und `p2`) von denen jeder 15 Produkte generiert und 3 Konsumenten (`c1`, `c2`, und `c3`) von denen jeder 10 Produkte konsumiert.

Die Buffergröße soll als Kommandozeilenparameter übergeben werden.

#### `semaphore.h`

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

typedef struct {
    int ctr;
    int wait;
    pthread_cond_t cond;
    pthread_mutex_t mutex;
} semaphore;

void sem_init(semaphore *sem, int ctr);

void up(semaphore *sem);

void down(semaphore *sem);
```

#### `semaphore.c`

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

#include "semaphore.h"

void sem_init(semaphore *sem, int ctr) {
    sem->ctr = ctr;
```

```

sem->wait = 0;
pthread_cond_init( &(sem->cond), NULL );
pthread_mutex_init( &(sem->mutex), NULL );
}

////////////////////////////////////////////////////////////////

void down(semaphore *sem) {
pthread_mutex_lock( &(sem->mutex) );
if (sem->ctr > 0) { sem->ctr -=1; }
else {
sem->wait +=1;
pthread_cond_wait( &(sem->cond), &(sem->mutex) );
}
pthread_mutex_unlock( &(sem->mutex) );
}

////////////////////////////////////////////////////////////////

void up(semaphore *sem) {
pthread_mutex_lock( &(sem->mutex) );
if (sem->wait == 0) sem->ctr +=1;
else {
sem->wait -=1;
pthread_cond_signal( &(sem->cond) );
}
pthread_mutex_unlock( &(sem->mutex) );
}

```

**Beispielausgabe: für \$ ./pkp 3**

```

p2: insert Item (buffer fill level: 1)
p2: insert Item (buffer fill level: 2)
c2: remove Item (buffer fill level: 1)
c1: remove Item (buffer fill level: 0)
p2: insert Item (buffer fill level: 1)
p1: insert Item (buffer fill level: 2)
p2: insert Item (buffer fill level: 3)
...
p2: insert Item (buffer fill level: 1)
p2: insert Item (buffer fill level: 2)
c3: remove Item (buffer fill level: 1)
c2: remove Item (buffer fill level: 0)

```