

# Übungen zu Systemnahe Programmierung in C (SPiC)

Peter Wägemann, Heiko Janker, Moritz Strübe, Rainer Müller  
(Lehrstuhl Informatik 4)



Wintersemester 2014/2015



## Aufgabe: tbsh

- nebenläufig veränderte Variable auch unter Linux: `volatile`
- Signale blockieren: `sigprocmask()`
- Signalhandler installieren: `sigaction()`
- Kein `sigprocmask(SIG_BLOCK, ...)` für SIGINT im Vater
- Blockieren von Signalen ist keine Behandlung von Signalen



## Inhalt

Aufgabe: tbsh

Theorieüberblick

Entwicklungsmuster der Mikrocontroller-Programmierung



## Aufgabe: tbsh – 4 Konzepte I

### 1. Signalhandler installieren (`sigaction()`):

```
1 struct sigaction act;
2 act.sa_handler = SIG_DFL; // Handlersignatur: void f(int signum)
3 act.sa_flags = SA_RESTART;
4 sigemptyset(&act.sa_mask);
5 sigaction(SIGINT, &act, NULL);
```

### 2. Signale blockieren/deblockieren (`sigprocmask()`):

```
1 sigset_t set;
2 sigemptyset(&set);
3 sigaddset(&set, SIGUSR1);
4 sigprocmask(SIG_BLOCK, &set, NULL); /* Blockiert SIGUSR1 */
5 // kritischer Abschnitt
6 sigprocmask(SIG_UNBLOCK, &set, NULL); /* Deblockiert SIGUSR1 */
```



### 3. Neuen Prozess erstellen (fork())

```
1 pid_t p = fork();
2 switch(p) {
3     case -1: // Fehler - kein Kind
4         ...
5     case 0: // Kind
6         ...
7     default: // Vater
8         ...
9 }
```

### 4. Programm im aktuellen Prozess starten (exec())

```
1 char *args[4];
2 args[0] = "cp";
3 args[1] = "x.txt";
4 args[2] = "y.txt";
5 args[3] = NULL;
6 execvp(args[0], args);
```



[https://www4.cs.fau.de/Lehre/SS14/V\\_SPiC/Folien/V\\_SPiC\\_handout.pdf](https://www4.cs.fau.de/Lehre/SS14/V_SPiC/Folien/V_SPiC_handout.pdf)



## Bitoperationen

- 0-Setzen: `PORTB &= ~(1 << PB7)`
- 1-Setzen: `PORTB |= (1 << PB7)`
- Auffüllen: `... ~(0xff << (uint8_t)numLeds)`
- Bits abfragen: `if( setting & (1 << 3))...`



## Atomares Schlafenlegen

```
1 cli();
2 while(event != 0) {
3     sleep_enable();
4     sei();
5     sleep_cpu();
6     sleep_disable();
7     cli();
8 }
```

1. Schützen der Schleifenabbruchbedingung  
(aus beiden Richtungen in Kontrollfluss)
2. „Sicherheitsgatter“ `sleep_enable()/sleep_disable()`
3. `sei()` wird atomar mit dem nächsten Befehl ausgeführt
4. Interrupts wieder aktivieren



```
1 cli();  
2 uint16_t counter_copy = counter;  
3 sei();  
4 show_number(counter_copy);
```

- Lesen/Schreiben von 16-Bit Wert nicht atomar (8-Bit Mikrocontroller)
- Interrupt sperren bei Zugriff auf nebenläufig veränderte Variable
- Lokale Kopie verhindert längeres Sperren von Interrupts

