

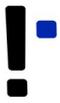
1. Worker-Threads vs. Worker-Prozesse

Betrachten Sie die Programme `worker-threads.c` (Listing 1, links) und `worker-prozesse.c` (Listing 2, rechts), die Sie auch von der Kurswebseite herunterladen können (`bs1-ue08.zip`).

```
1  #include <pthread.h> // pthread_*
2  #include <stdio.h>  // printf
3  #include <unistd.h> // sleep
4
5
6
7  #define MAX_WORKERS 10
8
9  int i;
10
11 void *worker (void *args) {
12     int i;
13     sleep (1);
14     char c = (int)args;
15     for (i=0; i<10; i++) {
16         printf ("%c", c); fflush (0);
17         sched_yield ();
18     }
19     return NULL;
20 }
21
22 int main () {
23     printf ("--Start--\n");
24     pthread_t threads[MAX_WORKERS];
25     int i;
26
27     // Threads erzeugen
28     for (i=0; i<MAX_WORKERS; i++) {
29         pthread_create (&threads[i],
30                        NULL,
31                        worker,
32                        (void*)(i+'A'));
33     }
34
35     // auf Threads warten
36     for (i=0; i<MAX_WORKERS; i++)
37         pthread_join (threads[i], NULL);
38
39     printf ("\n--Fertig--\n");
40     return 0;
41 }
42
43
44 #include <stdio.h> // printf
45 #include <unistd.h> // sleep
46 #include <sys/types.h> // waitpid
47 #include <sys/wait.h> // waitpid
48 #include <sched.h> // sched_yield
49
50 #define MAX_WORKERS 10
51
52 int i;
53
54 void *worker (void *args) {
55     int i;
56     sleep (1);
57     char c = (int)args;
58     for (i=0; i<10; i++) {
59         printf ("%c", c); fflush (0);
60         sched_yield ();
61     }
62     return NULL;
63 }
64
65 int main () {
66     printf ("--Start--\n");
67     int pids[MAX_WORKERS];
68     int i;
69
70     // Kindprozesse erzeugen
71     for (i=0; i<MAX_WORKERS; i++) {
72         pids[i] = fork ();
73         if (pids[i] == 0) {
74             // Sohn...
75             worker ((void*)(i+'A'));
76             return 0;
77         }
78     }
79
80     // auf Kindprozesse warten
81     for (i=0; i<MAX_WORKERS; i++)
82         waitpid (pids[i], NULL, 0);
83
84     printf ("\n--Fertig--\n");
85     return 0;
86 }
87
```

a) Lesen Sie zunächst die beiden Programme und überlegen Sie, welches Verhalten bei der Ausführung zu erwarten ist. (Zur Erklärung von zwei vermutlich unbekanntenen Funktionen:

- Die Funktion `sched_yield()` in Zeile 17 aktiviert den Scheduler, der aufrufende Prozess bzw. Thread gibt damit freiwillig die CPU ab.
- Die Funktion `fflush()` in Zeile 15 sorgt dafür, dass die mit `printf()` ausgegebenen Zeichen sofort im Terminal erscheinen und nicht gepuffert werden.)



2. Threads und fork

In dieser Aufgabe testen Sie, wie es sich auswirkt, wenn Sie `pthread_create()` und `fork()` gemeinsam verwenden.

a) Schreiben Sie ein kleines Testprogramm, das zunächst mit `fork()` den aktuellen Prozess verdoppelt. Vater- und Kindprozess erzeugen anschließend beide mit `pthread_create()` jeweils einen neuen Thread; der Vater führt darin die Funktion `vater()` aus, das Kind die Funktion `kind()`:

```
void *vater () {
    printf ("Neuer Thread im Vaterprozess\n");
    for (;;) ;
}
void *kind () {
    printf ("Neuer Thread im Kindprozess\n");
    for (;;) ;
}
```

Geben Sie auch vor und nach der Thread-Erzeugung eine kurze Meldung à la „Vater-Prozess vor `pthread_create`“ aus.

Starten Sie das Programm, betrachten Sie die Ausgabe und prüfen Sie (in einem anderen Terminalfenster) mit `ps -eLf`, welche Prozesse und Threads zum laufenden Programm gehören.

b) Jetzt ändern Sie die Reihenfolge: Das zweite Testprogramm erzeugt zunächst mit `pthread_create()` einen neuen Thread. Der neue Thread ruft dann `fork()` auf und beendet sich mit `return`, in der `main()`-Funktion warten Sie mit `pthread_join()` auf dieses Ereignis. Finden Sie auch hier heraus, was passiert – um die Prozesse und Threads mit `ps` beobachten zu können, bauen Sie an geeigneten Stellen wieder eine Endlosschleife `for (;;);` ein.

c) Hausaufgabe: Lesen Sie den Artikel „Threads and `fork()`: think twice before mixing them“ (über den Moodle-Kurs als PDF-Datei verfügbar) – er enthält eine Erklärung für das in Aufgabe **b)** beobachtete Verhalten.