



Bitte bearbeiten Sie die Aufgaben auf Ihrem Notebook in der virtuellen Linux-Maschine (Debian Linux, aus Datei SWF-Debian-2016.ova).

1. Worker-Threads vs. Worker-Prozesse

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

```
1 // Listing 1: worker-threads.c
2 #include <pthread.h> // pthread_*
3 #include <stdio.h> // printf
4 #include <unistd.h> // sleep
5
6 #define MAX_WORKERS 10
7
8 int i;
9
10 void *worker (void *args) {
11     int i;
12     sleep (1);
13     char c = (int)args;
14     for (i=0; i<10; i++) {
15         printf ("%c", c); fflush (0);
16         sched_yield ();
17     }
18     return NULL;
19 }
20
21 int main () {
22     printf ("--Start--\n");
23     pthread_t threads[MAX_WORKERS];
24     int i;
25
26     // Threads erzeugen
27     for (i=0; i<MAX_WORKERS; i++) {
28         pthread_create (&threads[i],
29             NULL,
30             worker,
31             (void*)(i+'A'));
32     }
33
34     // auf Threads warten
35     for (i=0; i<MAX_WORKERS; i++)
36         pthread_join (threads[i], NULL);
37
38     printf ("\n--Fertig--\n");
39     return 0;
40 }
41
42 }
```

```
1 // Listing 2: worker-prozesse.c
2
3 #include <stdio.h> // printf
4 #include <unistd.h> // sleep
5
6 #define MAX_WORKERS 10
7
8 int i;
9
10 void *worker (void *args) {
11     int i;
12     sleep (1);
13     char c = (int)args;
14     for (i=0; i<10; i++) {
15         printf ("%c", c); fflush (0);
16         sched_yield ();
17     }
18     return NULL;
19 }
20
21 int main () {
22     printf ("--Start--\n");
23     int pids[MAX_WORKERS];
24     int i;
25
26     // Kindprozesse erzeugen
27     for (i=0; i<MAX_WORKERS; i++) {
28         pids[i] = fork ();
29         if (pids[i] == 0) {
30             // Sohn...
31             worker ((void*)(i+'A'));
32             return 0;
33         }
34     }
35
36     // auf Kindprozesse warten
37     for (i=0; i<MAX_WORKERS; i++)
38         waitpid (pids[i], NULL, 0);
39
40     printf ("\n--Fertig--\n");
41     return 0;
42 }
```

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 16 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.)

