

Bitte bearbeiten Sie die Aufgaben auf Ihrem Notebook in der virtuellen Linux-Maschine (Debian Linux, aus der Datei SWF-Debian-2016.ova für VirtualBox oder SWF-Debian-2018.ova für VMware).

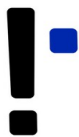
## 2. Worker-Threads vs. Worker-Prozesse

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
35
36     // auf Threads warten
37     for (i=0; i<MAX_WORKERS; i++)
38         pthread_join (threads[i], NULL);
39
40     printf ("\n--Fertig--\n");
41     return 0;
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 unbekannten 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.)



**b)** Laden Sie in einem Terminalfenster mit

```
wget swf.hgesser.de/vb-b1-ss2017/prakt/worker.zip
```

die beiden C-Quellcode-Dateien (die von der Kursseite auf [swf.hgesser.de](http://swf.hgesser.de) aus verlinkt sind) in der virtuellen Linux-Maschine herunter, entpacken Sie das Archiv mit

```
unzip worker.zip
```

und übersetzen Sie die Programme mit den folgenden Befehlen:

```
gcc -pthread -o worker-threads worker-threads.c  
gcc -o worker-prozesse worker-prozesse.c
```

Führen Sie dann beide Programme mehrfach aus. (Dem Programmnamen stellen Sie immer ./ voran, damit die Shell das Programm im aktuellen Ordner findet.) Sie sollten Ausgaben erhalten, die den folgenden ähneln:

```
student@swfdebian:~/c$ ./worker-threads  
--Start--  
DEFGHIJCBCFIGHJECBIFHGJECBIFGHJECBFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGEDDDDDDDAAAAA  
--Fertig--  
student@swfdebian:~/c$ ./worker-prozesse  
--Start--  
DEFGHIJCCIGJFHECIGFJHECIGFJHECIGFJHECIGFJHECIGFJHECIGFJHECIGFJHECIGFJHECIGFJHEDDDDDDDBBBBBBBBBAAAAA  
--Fertig--
```

(Ziehen Sie das Terminalfenster am besten so breit, dass die Ausgabe zwischen --Start-- und --Fertig-- ohne Zeilenumbruch angezeigt wird.)

**c)** Kommentieren Sie in beiden Quellcode-Dateien die erste Zeile aus der Funktion `worker()` (Zeile 11) durch Voranstellen von `//` aus. Dann ist `i` nicht länger eine lokale Variable der Funktion, und die `for`-Schleife verwendet die globale Variable `i` (aus Zeile 8) als Schleifenvariable.

Übersetzen Sie beide Programme nach dieser Änderung neu und führen Sie mehrere Testläufe durch.

– Welche Veränderung können Sie beobachten,

– und was verursacht diese Veränderung?

**d)** Passen Sie in beiden Programmen die Funktion `worker()` an, indem Sie vor dem `return`-Befehl in Zeile 18 noch einen Aufruf `sleep(30)`; einbauen (welcher die Funktion 30 Sekunden warten lässt). Kompilieren Sie dann beide Programme neu, starten Sie diese und betrachten Sie in einem zweiten Terminalfenster die Prozessliste. Für die Prozess-Variante (`worker-prozesse`) verwenden Sie dazu die Kommandos

```
ps auxw | egrep "USER|worker-"
```

und

```
pstree -p | grep "worker-"
```

Bei der Thread-Variante (`worker-threads`) geben Sie den Befehl

```
ps -eLf | egrep "UID|worker-"
```

ein, um auch die Threads zu sehen. (Die `grep`- bzw. `egrep`-Befehle, die über `|` an die Aufrufe von `ps` bzw. `pstree` angehängt sind, filtern die Ausgabe und lassen nur die Zeilen erscheinen, die hier von Interesse sind.) Was bedeuten bei den Threads die Angaben in den Spalten `PID` und `LWP`? Mit `man ps` und `man pstree` rufen Sie die eingebaute Dokumentation zu `ps` und `pstree` auf.

**e)** Lassen Sie mit `ldd worker-prozesse` bzw. `ldd worker-threads` die Listen der von den Programmen verwendeten dynamischen Bibliotheken anzeigen – an der Ausgabe können Sie schon erkennen, welches davon multithreaded ist und welches nicht.