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.

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

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.)

Seite 2/2

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 <u>swf.hgesser.de</u> 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--
DEFGHIJCBCBFIGHJECBIFGHJEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGEBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJGBCFIHJ
```

--Fertig--(Ziehen Sie das Terminalfenster am besten so breit, dass die Ausgabe zwischen --Start-- und --

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,

Fertig-- ohne Zeilenumbruch angezeigt wird.)

- 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 1dd worker-prozesse bzw. 1dd 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.