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

3. Synchronisation mit Barrieren

Neben den Mutexen und Semaphoren, die Sie im Skript kennengelernt haben, gibt es noch weitere *Synchronisationsprimitive* – darunter die so genannte *Barriere*, die in ihrer Funktion am deutlichsten eine „synchronisierende" Wirkung hat.

Eine Barriere kann das folgende gewünschte Verhalten einer Menge von Threads erzwingen:

- Mehrere Threads arbeiten gemeinsam an einer Aufgabenstellung.
- Jeder Thread arbeitet prinzipiell unahängig von den übrigen Threads, aber die Arbeiten sind in jedem Thread in mehrere Abschnitte unterteilt.
- Ein Thread darf erst dann von einem Abschnitt in den nächsten wechseln, wenn auch alle übrigen Threads den Abschnitt komplettiert haben.

Dazu wird eine Barriere zwischen den Abschnitten errichtet. Hat ein Thread einen Abschnitt komplettiert, meldet er das durch Aufruf einer Barrieren-Funktion. Er wird an dieser Stelle blockieren und seine Arbeit erst dann fortsetzen, wenn auch alle übrigen Threads den Abschnitt komplettiert und ebenfalls die Barrieren-Funktion aufgerufen haben. Der letzte Thread aus der Gruppe kann seine Arbeit also ohne Unterbrechung fortsetzen und in den nächsten Abschnitt wechseln, und nach ihm können auch alle übrigen Threads in den nächsten Abschnitt wechseln.

```
1   // Listing 1: synchronized-workers.c
2
3   #include <pthread.h>
4   #include <sys/types.h>
5   #include <stdio.h>
6   #include <sys/syscall.h>
7
8   pthread_t         t1, t2, t3;
9   pthread_barrier_t barrier;   // Deklaration einer Barrieren-Variable
10
11  void *worker (void *args) {
12    int i, ret = 0;
13    int tid = syscall (__NR_gettid);  // Get thread ID
14    for (i=0; i<10; i++) {
15      printf ("[%d]: i=%d\n", tid, i);
16      if ((i==3) || (i==6)) {
17        printf ("Thread %d erreicht Barriere\n", tid);
18        // hier Platz für Barrieren-Code; setze ret
19        printf ("Thread %d hinter Barriere (ret=%d)\n", tid, ret);
20      }
21    }
22  }
23
24  int main () {
25    // hier Platz für Barrieren-Initialisierung
26    printf ("Start\n");
27    pthread_create (&t1, NULL, worker, NULL);
28    pthread_create (&t2, NULL, worker, NULL);
29    pthread_create (&t3, NULL, worker, NULL);
30    pthread_join (t1, NULL);
31    pthread_join (t2, NULL);
32    pthread_join (t3, NULL);
33    printf ("Fertig\n");
34  }
```

Betrachten Sie das Programm `synchronized-workers.c` (Listing 1), das Sie auch von der Kurswebseite (im Archiv `barriere.zip`) herunterladen können.

a) Lesen Sie das Programm. Zur Erklärung einiger interessanter Code-Abschnitte:

Es gibt zwar mit `getpid()` eine Funktion, die die Prozess-ID zurückgibt, die entsprechende Funktion `gettid()` für die Thread-ID fehlt aber in der Standardbibliothek. Linux stellt aber einen System Call für diese Abfrage bereit, der sich über `syscall(__NR_gettid)` aufrufen lässt. Das passiert in Zeile 13.

Das Hauptprogramm erzeugt in Zeile 27–29 drei Worker-Threads, welche alle dieselbe Funktion `worker()` ausführen.

Jeder Worker-Thread gibt i.W. zehn Zeilen Text aus, in denen jeweils die Thread-ID und die Schleifenvariable `i` stehen. Die Arbeit soll in drei Abschnitte (0..3, 4..6, 7..9) unterteilt werden, und die Übergänge von einem Abschnitt in den nächsten sollen über eine Barriere synchronisiert werden. Der nötige Code für die Synchronisation fehlt noch.

b) Laden Sie in in der Debian-VM in einem Terminalfenster mit

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

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

```
unzip barriere.zip
```

und übersetzen Sie das Programm mit dem folgenden Befehl:

```
gcc -pthread -o synchronized-workers synchronized-workers.c
```

Führen Sie das Programm dann mehrfach aus. (Dem Programmnamen stellen Sie immer `./` voran, damit die Shell das Programm im aktuellen Ordner findet.)

Sie erkennen dann, dass die Barriere (wie zu erwarten) noch nicht in Betrieb ist, denn die Ausgaben „… erreicht Barriere“ und „… hinter Barriere“ erscheinen immer direkt hintereinander, z. B. wie im nebenstehenden Block: Die Threads arbeiten einfach durch. Das gilt es nun zu beheben.

```
Start
[13764]: i=0
[13764]: i=1
[13764]: i=2
[13764]: i=3
Thread 13764 erreicht Barriere
Thread 13764 hinter Barriere (ret=0)
[13764]: i=4
[13764]: i=5
...
```

c) Installieren Sie in der Debian-VM mit dem Kommando

```
sudo apt-get install manpages-posix-dev
```

eine umfangreichere Sammlung von Manpages nach. Darunter befinden sich nun auch die Manpages für `pthread_barrier_init()` und `pthread_barrier_wait()`. Lesen Sie diese mit

```
man pthread_barrier_init
man pthread_barrier_wait
```

durch. Falls die Nachinstallation nicht gelingt, finden Sie die Manpages auch online:

https://linux.die.net/man/3/pthread_barrier_init
https://linux.die.net/man/3/pthread_barrier_wait

d) Bauen Sie sinnvolle Aufrufe der beiden Funktionen in das Programm ein und testen Sie, ob es jetzt die Barrieren-Funktionalität umsetzt.

e) Passen Sie das Programm so an, dass es mit einer beliebigen Zahl von Worker-Threads arbeitet, die vor Zeile 8 über

```
#define NUMBER_OF_WORKERS 5
```

definiert (im Beispiel als 5) wird. Verwenden Sie dazu ein `thread_t`-Array und denken Sie daran, dass die Barriere die Zahl der zu synchronisierenden Threads kennen muss.