Kurzübersicht: POSIX Threads

	pthread-Funktion	*VL-Pendant*	*Beschreibung*
Thread-Management	`pthread_create(threadid, attr, start_routine, arg)`	`fork`	Erstellt und startet einen neuen Thread. Die als Funktionszeiger übergebene `start_routine` gibt die parallel auszuführende Funktion an. Mit den optionalen Parametern `attr` und `arg` können zusätzliche Eigenschaften des Threads bzw. ein Argument angegeben werden, das `start_routine` übergeben werden soll.
	`pthread_join(threadid, status)`	`join`	Blockiert den aufrufenden Thread bis der durch `threadid` repräsentierte Thread beendet ist. Wurde als `status` ein Zeiger ungleich `NULL` übergeben, wird an dessen Ziel der `pthread_exit()` übergebene Wert geschrieben. Gegebenenfalls muss der Thread, auf den gewartet werden soll, via `pthread_attr_setdetachstate()` auf `JOINABLE` gesetzt werden.
	`pthread_exit(status)`	`end`	Beendet den aufrufenden Thread und leitet das übergebene Argument an einen optionalen Aufruf von `pthread_join()` weiter. Das Programm wird beendet, wenn alle Threads sich beendet haben oder der gesamte Prozess via `exit()` beendet wurde. **Unterschiede zur VL:** Erlaubt die Übergabe eines Ergebniswertes
Sperren	`pthread_mutex_init(mutex, attr)`		Initialisiert das übergebene Mutex-Objekt (engl. **mut**ual **ex**clusion, gegenseitiger Ausschluss). Der Parameter `attr` ist optional.
	`pthread_mutex_destroy(mutex)`		Gibt die vom übergebenen Mutex-Objekt belegten Ressourcen frei.
	`pthread_mutex_lock(mutex)`	`lock`	Sperrt das angegebene Mutex-Objekt. Falls das Objekt zum Zeitpunkt des Aufrufs bereits gesperrt ist, wird der aufrufende Thread blockiert, bis das Mutex-Objekt wieder verfügbar wird.
	`pthread_mutex_unlock(mutex)`	`unlock`	Gibt die Sperre des übergebenen Mutex-Objektes wieder frei.
Signale	`pthread_cond_init(condition, attr)`		Initialisiert das übergebene Condition-Objekt. Der Parameter `attr` ist optional.
	`pthread_cond_destroy(condition)`		Gibt die vom übergebenen Condition-Objekt belegten Ressourcen frei.
	`pthread_cond_signal(condition)`	`signal`	Deblockiert mindestens einen der auf das Signal des übergebenen Condition-Objekts wartenden Threads. Falls zum Zeitpunkt des Aufrufs kein Thread auf das Signal wartet wird es verworfen. Der Aufruf sollte durch eine Sperre des selben Mutex-Objektes wie

Signale			das entsprechende `pthread_cond_wait()`-Pendant gesichert sein. **Unterschiede zur VL:** u.U. kann ein einziger Aufruf mehrere Threads wecken; Signale werden nicht gespeichert
	`pthread_cond_broadcast(condition)`		Deblockiert alle auf das Signal der übergebenen Condition wartende Threads.
	`pthread_cond_wait(condition, mutex)`	`wait`	Blockiert den aufrufenden Thread bis zur Signalisierung via `pthread_cond_signal()` oder `pthread_cond_broadcast()`. Das angegebene Mutex-Objekt muss gesperrt sein, wird zum Warten freigegeben und vor der Rückkehr wieder gesperrt. Da ein Aufwachen auch ohne ein explizites Signal möglich ist, sollte nach jedem Aufwachen die zu erwartende Bedingung erneut geprüft werden. **Unterschiede zur VL:** vorheriges `pthread_mutex_lock()` notwendig; spurious wakeups möglich

Eine vollständige Beschreibung der Funktionen finden Sie durch Klick auf den Funktionsnamen oder auf den man-Pages mittels `man <funktionsname>`.

Beispiel zu Verwendung von POSIX Threads:

```c
#include <stdlib.h>
#include <stdbool.h>
#include <pthread.h>

pthread_mutex_t mutex;
pthread_cond_t cond;
bool worker_finished;

void *workerThread(void *arg)  {
    doWork();

    /* Ende signalisieren */
    pthread_mutex_lock(&mutex);
    worker_finished = true;
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&mutex);

    pthread_exit(NULL);
}

int main(int argc, char **argv) {
    pthread_t thread;
    worker_finished = false;

    /* Initialisierung und Worker-Thread starten */
    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&cond, NULL);
    pthread_create(&thread, NULL, workerThread, NULL);

    doSomethingElse();

    /* Ende des Worker-Threads abwarten */
    pthread_mutex_lock(&mutex);
    while (!worker_finished) {
        pthread_cond_wait(&cond, &mutex);
    }
    pthread_mutex_unlock(&mutex);

    exit(EXIT_SUCCESS);
}
```