21. Thread-Funktion ohmthread_join()

Auf der Webseite finden Sie das Aufgaben-Archiv `sp-ss2013-ue12.tgz`.

Beim Nachbilden der Standardfunktionen aus der `pthread`-Bibliothek fehlt uns noch die Funktion `ohmthread_join()`, die Sie nun analog zu `pthread_join()` implementieren. Letztere Funktion hat folgende Signatur:

```
int pthread_join(pthread_t thread, void **value_ptr);
```

Sie nimmt also als erstes Argument eine Thread-Variable und als zweites einen Zeiger auf einen `void`-Zeiger. Dabei funktioniert das Ganze wie folgt: Wenn es eine Variable `void *ret` gibt und sich der Thread, auf den Sie warten wollen, mit `ohmthread_exit (ret)` beendet hat, dann soll dieser Rückgabewert über das zweite Argument von `ohmthread_join()` erreichbar sein. Dazu rufen Sie die Funktion wie folgt auf, wenn es sich z. B. um einen Integer-Wert handelt:

```
void *ret;
ohmthread_join (thr1, &ret);
printf ("Rueckgabewert: %d\n", *(int *)ret);
```

Die Datei `21/pthr-join.c` im Aufgabenarchiv enthält Beispielcode, der die `pthread`-Bibliothek verwendet und die Rückgabewerte von zwei Threads auswertet (testen Sie das Programm, gcc braucht die Option `-lpthread`) – dasselbe soll nachher auch mit `ohmthread_join()` funktionieren. (Bei der Ausgabe muss man `ret` erst in einen `(int *)` casten und dann mit `*` auf den Wert zugreifen.)

Ein Thread, der `ohmthread_join()` aufruft, wartet auf ein Ereignis – er darf also vom Scheduler nicht mehr aufgerufen werden, und außerdem muss der Prozess direkt nach dem (genauer: beim) Aufruf der Funktion zu einem anderen Thread wechseln. Dafür benötigen wir einen neuen Thread-Zustand

```
#define THREAD_WAIT          3
```

Der Scheduler soll dann bei der Auswahl des nächsten Threads solche im Wartezustand überspringen (genau wie er das bisher für Threads im Zustand `THREAD_EXIT` macht).

Um unmittelbar (ohne auf den Alarm zu warten) in den Scheduler zu springen, können Sie von Hand ein `SIGALRM`-Signal an den Prozess schicken; dazu verwenden Sie den Aufruf

```
raise (SIGALRM);
```

Im Thread-Control-Block des wartenden Threads müssen Sie außerdem eintragen, auf welchen anderen Thread dieser wartet. Dazu ergänzen Sie in `struct tcb` ein Element `pthread_t waitfor`.

Das Ganze kann nur funktionieren, wenn die Funktion `pthread_exit()` die Threadliste nach Threads durchsucht, welche auf den gerade zu beendenden Thread warten – als letzte Aktivität muss `pthread_exit()` also in allen auf `tid` wartenden Threads den Status von `THREAD_WAIT` auf `THREAD_ACTIVE` zurücksetzen, damit diese wieder vom Scheduler berücksichtigt werden.

Wird ein derart reaktivierter Thread fortgesetzt, steckt er ja noch mitten in der Abarbeitung der Funktion `ohmthread_join()` (die ihn vorher schlafen gelegt hatte) – das ist die Gelegenheit, um aus dem TCB des beendeten Threads dessen Rückgabewert auszulesen und dann den Thread aus der Threadliste zu streichen. (Anderenfalls würde er im „Zombie“-Zustand `THREAD_EXIT` in der Liste verbleiben.)

Eine weitere Änderung ist nötig: Die Funktion `initialize_threading()`, die beim Erzeugen des ersten Threads aufgerufen wird, muss am Ende `raise (SIGALRM)` ausführen, damit direkt in den Scheduler gesprungen wird – anderenfalls kann es passieren, dass das Hauptprogramm `thread_join()` aufruft, bevor das Threading überhaupt aktiviert wurde.

Die Funktion `ohmthread_join()` muss auch den Fall berücksichtigen, dass der Thread, auf den sie warten soll, bereits beendet (im Zustand `THREAD_EXIT`) ist – in dem Fall kann direkt der Rückgabewert ausgelesen werden, und der aufrufende Thread arbeitet ohne Verzögerung (und ohne Sprung in den

Scheduler) weiter.

Falls Sie Ihr Programm `21-threads-08.c` nennen, können Sie das `Makefile` aus dem Aufgabenarchiv (im Ordner `21/`) zum Kompilieren nutzen; Sie benötigen dazu außerdem die Assembler-Datei `call_ohmthread_exit.s` aus der letzten Übung, die ebenfalls in diesem Ordner liegt.

Passen Sie `main()` so an, dass mindestens zwei Threads erzeugt werden, die verschiedene Werte mit `return` zurückgeben; warten Sie dann in `main()` mit `ohmthread_join()` auf diese und geben Sie die jeweiligen Rückgabewerte aus.

22. ohmthread-Bibliothek

Im letzten allgemeinen Schritt (bevor es unterschiedliche Aufgaben für Teilgruppen gibt) machen wir aus den `ohmthread`-Funktionen eine Bibliothek – damit können Sie dann später `#include "ohmthread.h"` in ein Programm schreiben und es mit `gcc program.c ohmthread.o call_ohmthread_exit.o` (ähnlich zu `gcc program.c -lpthread`) übersetzen.[1]

Kopieren Sie Ihre Variablendeklarationen, Konstanten und Thread-Funktionen in eine neue Datei `ohmthread.c` und erstellen Sie zusätzlich eine Header-Datei `ohmthread.h`, welche die Signaturen der „öffentlichen“ Thread-Funktionen und die öffentlichen Konstanten enthält; im Aufgabenarchiv finden Sie im Ordner `22/` ein Beispiel für die Datei `ohmthread.h` (aus der Musterlösung), die Sie evtl. an Ihren Code anpassen müssen.

Die Datei `ohmthread.c` soll auch am Anfang `ohmthread.h` inkludieren, so dass Sie die Definitionen nicht zweimal schreiben müssen. Für im aktuellen Ordner liegende Header-Dateien geht das mit `#include "..."` (und nicht `#include <...>`). Übersetzen Sie dann probeweise die C-Datei mit

```
gcc -c ohmtread.c
```

(Das erzeugt eine neue Datei `ohmthread.o`.)

Entfernen Sie noch den Strg-C-Handler und dessen Initialisierung aus der Thread-Bibliothek; dieser war nur für Testzwecke gedacht.

Kopieren Sie den Rest (die Funktion `main()` sowie die Thread-Funktionen, aber auch alle `#include`-Befehle aus `ohmthread.c`) in die neue Programmdatei `22-threads-09.c`. Sie können dann das Makefile

```
22-threads-09: 22-threads-09.c ohmthread.o call_ohmthread_exit.o
	gcc 22-threads-09.c ohmthread.o call_ohmthread_exit.o -o $@
%.o: %.s
	gcc -c $<
%.o: %.c
	gcc -c $<
```

(das auch im Unterordner `22/` liegt) verwenden, um die Gesamtanwendung zu bauen; sie sollte sich genauso verhalten wie Ihr Programm aus Aufgabe 21.

Die Datei `ohmthr-join.c` enthält eine auf `ohmthread`-Funktionen „portierte“ Version des Testprogramms aus `pthr-join.c` – stellen Sie sicher, dass dieses korrekt (also wie die `pthread`-Version) arbeitet. Sie sollten es mit

```
gcc ohmthr-join.c ohmthread.o call_ohmthread_exit.o
```

kompilieren können.

Damit ist das erste Zwischenziel erreicht: Thread-Programme, die nur `pthread_create()`, `pthread_exit()` und `pthread_join()` verwenden, können Sie mit geringem Aufwand auf unsere `ohmthread`-Bibliothek portieren. Die Signaturen der `*_exit()`- und `*_join()`-Funktionen sind gleich, bei `ohmthread_create()` fehlt im Vergleich zu `pthread_create()` nur deren zweites Argument (die Optionen für das Erzeugen eines Threads).

Nächste Woche geht es mit verschiedenen Projektaufgaben (u. a. Scheduler, Synchronisierung) weiter.

---

1 Eine echte Bibliothek (die Sie beim `gcc`-Aufruf mit `-lohmthread` linken könnten) entsteht auf diese Weise nicht, es geht hier nur um das Prinzip der Trennung von Bibliotheks- und Programmcode.