2 Thread-Koordination: Die trainierenden PhilosophInnen

2. 1 Vorübung: Makefile

Das Programm `critsect02.c` aus meinem Pub-Verzeichnis zeigt den Schutz kritischer Abschnitte von Threads mit Semaphoren. Das Übersetzen von Hand erfolgt mit dem Befehl

```
/usr/bin/gcc -pthread -g -Wall -lpthread -o critsect02 critsect02.c
```

Aufgabe: Schreiben Sie ein Makefile, das den Kompiliervorgang mit dem Befehl `make critsect02` ausführt

2. 2 Thread-Koordination in der Muckibude

2. 2.1 Problemstellung

Die PhilosophiestudentInnen Anna, Bernd, Clara, Dirk und Emma gehen nach dem Seminar zum Training ins Sportstudio, denn ein klarer Geist benötigt einen starken Bizeps. Die Kraftmaschinen müssen mit Gewichten entsprechend dem individuellen Trainingsplan bestückt werden. Der sieht für unsere Gruppe so aus:

Anna	6 kg
Bernd	8 kg
Clara	12 kg
Dirk	12 kg
Emma	14 kg

Im Magazin befinden sich folgende Gewichte:

4	Gewichte zu	2 kg
4	Gewichte zu	3 kg
5	Gewichte zu	5 kg

Der Ablauf im Sportsudio besteht im fortwährend wiederholten Zyklus

- Gewichte holen: `GET_WEIGHTS`
- Trainieren: `WORKOUT`
- Gewichte wegbringen: `PUT_WEIGHTS`
- Ausruhen: `REST`

Wenn die laut Trainingsplan nötigen Gewichte nicht da sind, blockierten die jeweiligen SportlerInnen.

Aufgabe: Schreiben Sie ein Programm zur Simulation des Problems der trainierenden Philosophie-Studis. Die Koordination des Zugriffs auf die Gewichte soll mit einem *Monitor-Konzept* erfolgen, das Sie mit phtread-Mutexen und phtread-condvars realisieren. (Andere Lösungen werde ich nicht akzeptieren.)

- Alle PhilosophInnen werden durch dieselbe Threadfunktion `philothread` realisiert.
- Vor Eintritt in den Zustand `WORKOUT` durchläuft der Thread den Zustand `GET_WEIGHTS`, ruft die Monitor-Methode `get_weights` auf, in der er versucht, sich die nötigen Gewichte zu holen. Der Thread blockiert auf einer *condition variable*, wenn sich das laut Trainingsplan erforderliche Gesamtgewicht sich mit den verfügbaren Gewichten nicht realisieren lässt.
- Am Ende des Zustands *WORKOUT* durchläuft der Thread den Zustand `PUT_WEIGHTS`, in der die Gewichte ins Magazin zurückgegeben werden, und alle(!) blockierten Threads mit einem `phtread_cond_signal` geweckt werden.
- Die Tätigkeiten Trainieren und Ausruhen werden durch zwei (fast leere) Zählschleifen simuliert, in denen lediglich die Steuerbefehle ausgelesen und ggf. ausgeführt werden. Vorschlag für die Grenzwerte der Schleifen:

```
#define REST_LOOP 1000000000
#define WORKOUT_LOOP    500000000
```

- Damit man sieht, was passiert, soll *innerhalb des Monitors* bei jedem Zustandswechsel eines Threads eine Funktion `display__status` aufgerufen werden, die etwa so eine Ausgabe erzeugt:

```
0(6)n:R:[0, 0, 0] 1(8)n:G:[0, 0, 0] 2(12)n:R:[0, 0, 0] 3(12)n:R:[0, 0, 0] 4(14)n:R:[0, 0, 0]   Supply: [4, 4, 5]
0(6)n:R:[0, 0, 0] 1(8)n:W:[0, 1, 1] 2(12)n:G:[0, 0, 0] 3(12)n:R:[0, 0, 0] 4(14)n:R:[0, 0, 0]   Supply: [4, 3, 4]
0(6)n:R:[0, 0, 0] 1(8)n:W:[0, 1, 1] 2(12)n:W:[1, 0, 2] 3(12)n:G:[0, 0, 0] 4(14)n:R:[0, 0, 0]   Supply: [3, 3, 2]
0(6)n:R:[0, 0, 0] 1(8)n:W:[0, 1, 1] 2(12)n:W:[1, 0, 2] 3(12)n:W:[1, 0, 2] 4(14)n:G:[0, 0, 0]   Supply: [2, 3, 0]
0(6)n:G:[0, 0, 0] 1(8)n:W:[0, 1, 1] 2(12)n:W:[1, 0, 2] 3(12)n:W:[1, 0, 2] 4(14)n:G:[0, 0, 0]   Supply: [2, 3, 0]
0(6)n:W:[0, 2, 0] 1(8)n:W:[0, 1, 1] 2(12)n:W:[1, 0, 2] 3(12)n:P:[1, 0, 2] 4(14)n:G:[0, 0, 0]   Supply: [2, 1, 0]
0(6)n:W:[0, 2, 0] 1(8)n:W:[0, 1, 1] 2(12)n:W:[1, 0, 2] 3(12)n:R:[0, 0, 0] 4(14)n:G:[0, 0, 0]   Supply: [3, 1, 2]
0(6)n:W:[0, 2, 0] 1(8)n:W:[0, 1, 1] 2(12)n:P:[1, 0, 2] 3(12)n:R:[0, 0, 0] 4(14)n:W:[2, 0, 2]   Supply: [1, 1, 0]
```

- Hinter dem Thread-Index (0 bis 4) steht der Gewichtswert laut Trainingsplan, dann der Befehlsstatus: (n = normal, b = blockiert, q = quit), danach der Zustand im Trainings-Zyklus: (R = rest, G = get_weights, W = workout, P = put_weights), dann die Anzahl der zugeteilten Gewichte der drei Typen (2 kg, 3 kg, 5 kg).
- Diese Displayfunktion soll Alarm schlagen, wenn die Summe der zugeteilten Gewichte ungleich der Summe der insgesamt vorhandenen Gewichte ist, weil dann etwas mit der Synchronisation schief gegangen ist.
- Ihr `main`-Programm soll zunächst alle Synchronisationsobjekte erzeugen, dann soll es auf Tastatureingaben (mit `fgets`) warten und einen entsprechenden Befehlscode in ein globales Arrayelement schreiben, aus dem der zugeordnete Philosophen-Thread liest. Folgende Eingaben sollen möglich sein:

q | Q Benachrichtigen aller Philosophen-Threads, dass sie sich beenden sollen, warten mit `pthread_join`, dann löschen aller Synchronisationsobjekte, Programmende.

<philo_id>b Blockiere Philosophen-Thread <philo_id>. Dabei ist <philo_id> *eine* Ziffer zwischen 0 und `4`. Im Einzelnen: Schreibe den Befehlscode `'b'` in das Befehls-Array-Element des Philosophenthreads. Dieser führt in seiner Zählschleife in der `REST`- und `WORKOUT`- Methode ein `sem_wait` auf einem Semaphoren aus, der dem Philosophen gehört.

<philo_id>u Unblock. Führe `sem_post` auf dem entsprechenden Semaphoren aus und befreie dadurch den blockierten Philosophen-Thread.

<philo_id>p Proceed. Springe aus der Zählschleife ans Ende der Funktion `workout` bzw. `rest`.

- Bringen Sie die Monitor-Funktionen und -Daten in einem eigenen Quellfile unter, überlegen Sie sich, welche Headerfiles Sie anlegen wollen, und schreiben Sie ein Makefile, um Ihre Anwendung zu erzeugen.

Hinweis:

- Ihr Programm braucht nur für fünf PhilosophInnen zu funktionieren, nicht für eine beliebige Anzahl.
- Wenn Sie sportlichen Ehrgeiz haben, dürfen Sie die Kombinationen der Gewichte, die das Gesamtgewicht laut Trainingsplan darstellen, mit einem rekursiven Algorithmus ermitteln. Es gibt manchmal mehrere Möglichkeiten. 12 kg können beispielsweise durch $(2\cdot 5+1\cdot 2)$ kg oder durch $(4\cdot 3)$ kg realisiert werden. Sie dürfen die möglichen Kombination aber auch fest in Ihr Programm eintragen und in der Methode `get_weights` der Reihe nach durchprobieren.

Mens sana in corpore sano!