Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Betriebssysteme Praktikum 2

Entwurfsbeschreibung

Florian Heuer

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

1 Aufgabenstellung

1.1 Kurzbeschreibung

Im Praktikum 2 der Veranstaltung “Betriebssysteme” soll eine Simulation trainierender Philosophen in einer „Muckibude“ mit Hilfe von Threads, Semaphoren, Conditionvars und Mutexen (Monitorkonzept) realisiert werden. Dabei soll der Umgang mit Threads, insbesondere dessen Synchronisierung mit Werkzeugen des Betriebssystems erlernt werden.

2 Entwurf

Der Entwurf für das Programm „**MuscleFactory**“ sieht einen ähnlichen Ansatz wie in einer objektorientierten Sprache vor. In der eingesetzten Sprache C ist dies nicht ohne Weiteres möglich. Eine strukturierte Weise mit gekapselten Zuständigkeiten wird angestrebt und so wird der Code über mehrere dedizierte Sourcefiles aufgeteilt auf diese in 2.2 eingegangen wird. Zunächst wird die Umgebung beschrieben in der das Programm entwickelt wird.

2.1 Umgebung

Das Programm wird in Sprache C entwickelt. Als Compiler kommt „GCC“ zum Einsatz. Das Betriebssystem Open SUSE in Version 12.3 wird zum kompilieren und ausführen des Codes benutzt. Als Editor wird Sublime 2 verwendet. Zudem wird zur Erleichterung des Kompiliervorgangs ein Makefile erstellt.

2.2 Umsetzung

Wie oben erwähnt wird das Programm in verschiedene Files gekapselt. Diese werden unterhalb aufgeführt und relevante Details näher erläutert.

2.2.1 Global.h

Ein **Philosoph** wird als Struktur und eigener Typ abgebildet und hält dabei die Attribute für eine Thread ID, zu trainierendes Gewicht, Name, Modus (ob er blockiert, normal oder beendet ist), Status (ob er sich Gewichte holt, trainiert, Gewichte zurücklegt oder ruht), geholte Gewichte (Struktur mit 3 Integern repräsentativ für 2, 3, und 5kg) und eine Semaphore für die Blockierung des Threads.
Globale Variablen: Ein Array welches alle Philosophen (oben beschriebene Struktur) enthält. Ein Gewichtedepot, welches alle Gewichte der Muckibude enthält. Eine Conditionvariable

und ein Mutex zur Threadkoordinierung. Char Array zum speichern eines gelesenen Kommandos.
Funktionsprototypen: (get__status) Funktion zur Ausgabe des Zustandes der gesamten Muckibude. (getThreadID) Filterfunktion zum Auslesen der Thread ID aus einem übergebenen Komanndo.

2.2.2 Main.c

Implementationen der Funktionen get_status und getThreadID.
In der Main Methode werden alle Philosophen und Threads erzeugt. Ein Thread erhält dabei genau einen Pointer auf einen Philosohpen.
In einer While-Schleife werden danach auf Komandos mit fgets eingelesen und in dem globalen Char-Array gespeichert. Wird das Kommando „q“ eingelesen wird die Schleife beendet und auf alle Threads gewartet bis sich diese ebenfalls beendet haben. Das Programm endet. Das Kommando <id>u sorgt für die Freigabe eines blockierten Threads.

2.2.3 Philosoph.c

Hier wird die Threadfunktion implementiert, welche in der Main gestartet wird. Als parameter bekommt die einen Pointer auf eine Philosophenstruktur. Der Traningszyklus (Gewichte holen, trainieren, Gewichte zurücklegen und ruhen) eines Philsophen wird in der Funktion implementiert. Dabei ist zu beachten das hier der Eintritt in den kritischen Bereich mit Zugriff auf das Gewichtedepot erfolgt. Der kritische Bereich wird hier mit einer Conditionvariable und einem Mutex geschützt (Monitorkonzept).

2.2.4 MuscleFactory.c

In dieser Sourcedatei werden die Funktionen GET_WEIGHTS und PUT_WEIGHTS implementiert, welche für den Zugriff auf das Gewichtedepot der Muckibude zuständig sind. GET_WEIGHTS erhält dabei als einzigen Parameter den Philosopen, dieser hält das Trainingsgewicht welches vom Philosophen geholt werden möchte. Der Algorithmus entscheidet nun welche Kombination an Gewichten an den Philosophen gegeben wird bzw. ob dies möglich ist. Die Rückgabe ist 0 (Gewichte konnten nicht geholt werden, weil keine Kombination zum gewünschten Gewicht realisierbar ist.) oder 1 (Gewichtekombination wird an den Philosoph übergeben).

3 Abnahme

Der folgende Abschnitt soll alle vom Prüfer ausgehenden Anmerkungen bzw. geforderte Anpassungen am präsentierten Programm protokollieren.

3.1 Allgemein

Die Abnahme des Programms wurde erfolgreich am 22.11.2016 von Malte Nogalski im Labor 0761 durchgeführt. Im Zuge der Abnahme sind einige Änderungen am Code resultiert, welche im folgenden Punkt aufgeführt werden.

3.2 Anmerkungen & Änderungen

Die unterhalb aufgeführten Punkte beschreiben die getätigten Änderungen am Quellcode.

3.2.1 Eingabe & Befehlsauswertung

Die Eingabe der Befehle erfolgte nicht wie gewünscht nach dem Muster z.B. „2b“ oder „0u“ sondern so „<2>b“. Um die Befehle einzulesen war es nötig die spitzen Klammern mit anzugeben. Dies wurde während der Abnahme geändert.
Des Weiteren Funktionierte Befehl Proceed(z.B. „2p“), welcher an das Ende der Workout bzw. Restschleife springen soll nicht wie gewünscht. Das Problem war dabei, dass Permanent nach Eingabe an das Ende gesprungen wurde und nicht nur einen Trainingszyklus lang. Das Problem: der Befehl wurde nach dem Auslesen nicht zurückgesetzt. Dies wurde in der Abnahme behoben.

3.2.2 Makefile

Die Änderungen am Makefile umfassen stilistische Dinge. Die Funktion war gegeben. So z.B. waren in der Variable “SRC” nicht C Source Dateien angeben, sondern auch Objektfiles. Diese wurden dann in einer weiteren Variable „OBJ“ gespeichert. Mit den Headerfiles wurde auf gleiche Weise verfahren. Des Weiteren wurden redundante Targets zu einem generischen zusammen gefasst (siehe Code).

3.2.3 Philsophen-Thread

Der in der Methode philosoph() auftretende kritische Bereich wurde in eine dedizierte Methode ausgelagert an dessen Anfang und Ende, der Mutex lock bzw. unlock stehen, um klar zu machen, dass dies eine synchronisierte Methode ist, die einen kritischen Bereich betritt.

3.2.4 Beendigung aller Threads

Nach Eingabe des Quitbefehls lief das Programm nicht zu Ende, insofern das Programm noch geblockte Threads enthielt. Diese Blockierungen wurden nicht mit einem „sem_post“ auf die Semaphoren der Threads aufgehoben. Der Main Thread lief nun ewig weiter das dieser auf das „joinen“ der blockierten Threads wartete. Dies wurde in der Abnahme behoben.

Nach all diesen Änderungen funktionierte das Programm wie gewünscht und die Abnahme erfolgreich beendet.

4 Fazit

Insgesamt war diese Aufgabe eine kniffelige Angelegenheit, welche an manchen Stellen meiner Meinung nach nicht so klar definiert war, wie ich es mir gerne gewünscht hätte. Es hätte mich vor einigen Änderungen in der Abnahme bewahrt. Trotzdem hat es Spaß gemacht die Aufgabe zu lösen, da ich dadurch erst richtig verstanden habe, was Semaphoren, Mutexe und Conditionvars für die IPC bedeuten und wie man in C Konzepte wie z.B. „Monitor“ umsetzt.

5 Repository

Der komplette Sourcecode des Programms “**MuscleFactory**” ist unter folgendem Link auf GitHub zu finden.

https://github.com/FlowwX/MuscleFactory

6 Anhang

1. Quellcode (Zip-Archiv), Alternativ: oben genanntes Repository