Weitere Hinweise zu den Übungen Betriebssysteme

- Die abgegebenen Antworten/Programme werden automatisch auf Ähnlichkeit mit anderen Abgaben überprüft. Wer beim Abschreiben[1] erwischt wird, verliert ohne weitere Vorwarnung die Möglichkeit zum Erwerb der Studienleistung in diesem Semester!
- Die optionalen Aufgaben (davon wird es jeweils eine auf den Aufgabenblättern 1–3 geben) sind ein Stück schwerer als die „normalen“ und geben *keine* zusätzlichen Punkte für das jeweilige Aufgabenblatt – aber jeweils ein „Bonus-Sternchen“ ★. Wenn ihr alle drei Sternchen sammelt, müsst ihr das letzte Aufgabenblatt (A4) nicht bearbeiten!

Aufgabe 2: Threadsynchronisation (10 Punkte)

Theoriefragen (4 Punkte)

1. Erläutert den Unterschied zwischen aktivem und passivem Warten.
2. Was ist der Konvoi-Effekt? Bei welchen CPU-Zuteilungsstrategien kann dieser auftreten? Nennt ein Beispiel und erläutert warum bei der von euch angegebene Strategie der Konvoi-Effekt auftritt.
3. Folgende Prozesse erreichen die CPU-Warteschlange in dieser Reihenfolge: A mit Bedienzeit 10ms, B mit Bedienzeit 11ms, C mit Bedienzeit 1ms, D mit Bedienzeit 7ms und E mit Bedienzeit 3ms. Der Prozesswechsel dauert genau 0ms und ist somit zu vernachlässigen. Es gibt keine I/O-Bursts. Berechnet das Round-Robin-Scheduling. Die Zeitscheibe beträgt $t_{quantum}$=4ms. Gebt dazu den Startzeitpunkt sowie die Durchlaufzeit der Prozesse an.

Programmierung: Filmclub (6 Punkte)

Für die folgenden Aufgaben soll mit leichtgewichtigen Prozessen unter Linux (POSIX-Threads) gearbeitet werden, die ihr in den Übungen kennen lernen werdet. Ihr sollt folgendes Szenario im Laufe der Aufgabenbearbeitung erstellen:

Einige filmbegeisterte Studierende haben einen Filmclub als studentische AG gegründet. Um die Ausgaben und Einnahmen besser verwalten zu können, haben sie ein gemeinsames Konto eingerichtet, das sie selber gemeinsam verwalten. Da in einer studentischen AG viele Aufgaben anfallen, haben die Studierenden einzelne Teams gegründet, die sich um Aufgaben, wie Bestellung von Filmen oder das Kaufen von Getränken und Speisen kümmern. Da all diese Dinge bezahlt werden wollen, hat jedes Team Zugriff auf das gemeinsame Konto und kann darauf Transaktionen tätigen. Damit aber stets die Kontrolle über den Betrag auf dem Konto erhalten bleibt, notiert jedes Team den Kontostand vor und nach seiner Transaktion, so dass später eine Bilanz erstellt werden kann.

a) Vorbereitungen Erstellt zuerst eine globale Variable für den Kontostand und implementiert jeweils eine Funktion die Filme bestellt, eine Funktion die Getränke und Speisen einkauft und eine Funktion, die neues Geld auf das Konto überweist. Für die eigentliche Transaktion speichert ihr in jeder der Funktionen den globalen Kontostand in einer lokalen Variable zwischen und gebt diesen mittels printf() aus. Anschließend soll der zu überweisende Betrag auf den zuvor gemerkten Kontostand angerechnet und der neue Kontostand auf das globale Konto übertragen und wiederum ausgegeben

[1] Da wir im Regelfall nicht unterscheiden können, wer von wem abgeschrieben hat, gilt das für Original **und** Plagiat.

werden. Die Funktionen, die das Konto belasten (Filme bestellen bzw. Getränke und Speisen einkaufen), sollen ihren Funktionsrumpf jeweils fünf Mal ausführen. Die Funktion zum Geldeinzahlen soll den Rumpf nur zwei Mal ausführen.

Eine Film-Transaktion kostet 200 €, eine Getränke- bzw. Speise-Bestellung kostet 400 € und eingezahlt werden 2.000 €. Das aktuelle Guthaben des Kontos vor Programmstart beträgt 1.000 €.

Zunächst sollen die Funktionen über einfache Funktionsaufrufe aus main() aufgerufen werden. Nach dem alle Funktionen abgearbeitet wurden, soll in main() der Endkontostand ausgegeben werden. Speichert euer Programm in der Datei `filmclub_a.c`.

b) Threads erzeugen und starten Die Funktionen aus Aufgabenteil a) sollen nun parallel innerhalb einzelner Threads ausgeführt werden. Statt die drei Funktionen in main() aufzurufen müssen sie nun als *'start_routine'* an **pthread_create(3)** übergeben werden. Alle Threads sollen dabei auf das globale Konto unsynchronisiert zugreifen. Sorgt dafür, dass der Endkontostand nach Beendigung aller Threads (**pthread_join(3)**) ausgegeben wird.

Außerdem sollen sich von jetzt an die Threads nach dem Lesen des aktuellen Kontostands und seiner Ausgabe schlafen legen, bevor sie die Transaktion abschließen (Filme bestellen: 2 Sekunden, Getränke bzw. Speisen bestellen: 3 Sekunden, Neues Geld überweisen: 8 Sekunden). Speichert euer Programm dazu in der Datei `filmclub_b.c`

Wenn ihr die Ausgaben eures Programms betrachtet, stellt ihr wahrscheinlich fest, dass die Bilanz des Kontos am Ende einen Fehler aufweist. Beschreibt beispielhaft eine mögliche CPU-Zuteilung, die erklärt, dass eine Transaktion fehlerhaft abgewickelt werden kann (in `antworten.txt`).

c) Synchronisation Löst das Problem aus Aufgabenteil b) in `filmclub_c.c`, indem ihr den Zugriff auf das gemeinsame Konto synchronisiert. Nutzt dazu **pthread_mutex_init(3), pthread_mutex_lock(3)** und **pthread_mutex_unlock(3)**. Abschließend soll die Schlossvariable beim Beenden eures Programms mit **pthread_mutex_destroy(3)** wieder entfernt werden.

d) Sternchenaufgabe (optional)★ Bisher konnte es vorkommen, dass eine Transaktion nicht gedeckt war. In Zukunft soll vor jeder Transaktion, die Geld vom Konto abzieht, überprüft werden, ob das Konto dadurch überzogen wird. Jeder Thread, der eine nicht gedeckte Transaktion durchführen will, soll von nun an warten, bis genügend Geld auf dem Konto vorhanden ist (**pthread_cond_wait(3)**) und erst dann fortfahren. Nutzt dazu Condition Variables (**pthread_cond_init(3)**, **pthread_cond_signal(3)**) und erweitert euer Programm entsprechend in `filmclub_d.c`.
Tipp: Überlegt euch zuerst, welche eurer Threads auf Deckung ihrer Transaktionen warten müssen und welcher Thread den anderen mitteilt, dass ihre Transaktionen wieder gedeckt sind. Beachtet dabei, dass **pthread_cond_signal(3)** erst nach dem Aufruf von **pthread_cond_wait(3)** ausgeführt werden sollte.

Tipps zu den Programmieraufgaben:

- Kommentiert euren Quellcode ausführlich, sodass wir auch bei Programmierfehlern im Zweifelsfall noch Punkte vergeben können!
- Die Programme sollen dem ANSI-C- und POSIX-Standard entsprechen und sich mit dem gcc auf den Linux-Rechnern im FBI-Pool übersetzen lassen. Kompiliert euer Programm dazu mit den Compilerflags `-Wall -ansi -pthread`.[a]
- Alternativ kann auch der GNU C++-Compiler (*g++*) verwendet werden.

Abgabe: bis spätestens Freitag, 31. Mai 12:00 (Übungsgruppen 1/3/5/...) bzw. Dienstag, 4. Juni 12:00 (Übungsgruppen 2/4/6/...).

[a] Unabhängig davon ist das Compilerflag -Werror weiterhin empfehlenswert.