UNIVERSITÄT OSNABRÜCK

ESS

marcel.koeppen [☺] uni-osnabrueck.de
marcel.luetkedreimann [☺] uni-osnabrueck.de

Übung zur Vorlesung
Betriebssysteme
Wintersemester 2021/22

Programmierblatt 2: POSIX-Threads
Viren, Fäden und Knoten in der Abrechnung

➪ **Abgabe der Lösungen bis Montag, 29. November 14:00 im AsSESS**

Auf dem ersten Programmierblatt haben wir uns unter anderem mit UNIX-Prozessen beschäftigt. Für die folgenden Aufgaben sollen Sie POSIX-Threads (pthreads) verwenden. Dabei soll im Laufe der Aufgaben folgendes Szenario erstellt werden:

Durch die SARS-CoV-2 Pandemie mussten alle Planungen für das diesjährige Terrassenfest erneut verworfen werden. Das ORGA-Team arbeitet deswegen an einer Alternative: Es soll ein kleiner Weihnachtsmarkt auf dem Campusgelände stattfinden. Damit dabei die Übersicht über Ausgaben und Einnahmen behalten werden kann, wurde ein gemeinsames Konto eröffnet. Um die Vielfalt an anstehenden Aufgaben bewältigen zu können, hat sich das ORGA-Team in mehrere Arbeitsgruppen aufgeteilt. Eine Arbeitsgruppe, die Essen und Trinken besorgt, eine, die eine kleine Eisfläche und Schlittschuhe organisiert und eine weitere, die sich um sonstige Aufgaben, wie Weihnachtsbuden, Absperrungen, Toilettenwagen, etc. kümmert. Außerdem gibt es Leute, die für die Finanzierung, also das Sammeln von Spenden und Werbegeldern, zuständig sind. Jede dieser Arbeitsgruppen erhält Zugriff auf das gemeinsame Konto, da so ein unbürokratischer Ablauf der Beschaffungen gewährleistet ist. Damit am Ende trotzdem eine Abrechnung der einzelnen Kostenpunkte erfolgen kann, notieren sich die einzelnen Arbeitsgruppen den Kontostand vor und nach ihren Transaktionen. Zu Beginn hat das ORGA-Team 3000€ auf dem Konto.

Geben Sie Ihre Entwicklungsschritte jeweils in den angegebenen Dateien ab.

1 Vorbereitung (4 Punkte)

Erstellen Sie zunächst eine globale Variable für den Kontostand mit dem genannten Anfangsbetrag. Implementieren Sie anschließend für jede der vier Arbeitsgruppen eine eigene Funktion, die die notwendige Transaktion auf dem Konto durchführt.

- Verpflegungsgruppe: Hier sollen die Getränke und das Essen bestellt werden. Es ist also eine Abbuchung von 400€ zu tätigen.
- Eisfläche-Gruppe: Es soll eine kleine Eisfläche aufgebaut und Schlittschuh zur kostenfreien Ausleihe angeboten werden, daher ist auch hier eine Abbuchung von 600€ nötig.
- Sonstiges-Gruppe: Da diese Gruppe ein sehr breites Aufgabenspektrum hat, kann hier kein Pauschalbetrag überwiesen werden. Übergeben Sie dieser Funktion deshalb den zu buchenden Betrag als Parameter und buchen diesen anschließend jeweils ab.
- Finanzierungsgruppe: Es werden nur standardisierte Werbeflächen vergeben. Daher ist es hier ausreichend, wenn Sie einen konstanten Betrag von 2500€ zum Konto hinzu buchen.

Implementieren Sie die eigentliche Transaktion wie folgt: Zunächst wird der Kontostand in eine lokale Variable geladen und mittels **printf(3)** ausgegeben. Anschließend soll der Betrag mit dem lokal gespeicherten Kontostand verrechnet werden. Geben Sie den resultierenden Kontostand erneut aus und speichern ihn schließlich in der globalen Variablen.

Bei den Funktionen der Gruppen 'Verpflegung', 'Eisfläche' und 'Sonstiges' soll das in einer Schleife zehn Mal passieren, bei der Gruppe 'Finanzierung' nur fünf Mal. Für den Funktionsaufruf der Sonstiges-Gruppe können Sie sich selbst einen Parameter überlegen, der ins Budget passt.

Zunächst sollen die Funktionen über einfache Funktionsaufrufe aus main() aufgerufen werden. Nachdem alle Funktionen abgearbeitet wurden, soll in main() der Endkontostand ausgegeben werden.

Hinweise:

- Denken Sie an den Titel des Programmierblattes: POSIX-Threads. Sie haben bereits gesehen, dass Funktionen, die später durch eigene Threads ausgeführt werden sollen, der Signatur `void* func(void* args)` entsprechen müssen. Falls Sie einer solchen Funktion einen Parameter vom Typ `int` übergeben wollen, müssen Sie das über einen Zeiger auf eine entsprechende Variable realisieren. **Vorsicht:** Integer direkt als Pointer umzudeuten, funktioniert nicht plattformunabhängig!

→ `weihnachtsmarkt_a1.c`

2 Threads erzeugen und starten (3 Punkte)

Die Arbeitsgruppen wollen ihre Aktivitäten nun parallelisieren. Die Funktionen aus Aufgabe 1 sollen also parallel innerhalb einzelner Threads ausgeführt werden. Statt die vier Funktionen in main() aufzurufen, müssen sie nun als `'start_routine'` an **pthread_create(3)** übergeben werden. Am Ende von main() soll auf die Beendigung aller anderen Threads gewartet (**pthread_join(3)**) und anschließend der Endkontostand ausgegeben werden.

Da mittlerweile auch die Mitglieder des ORGA-Teams durch die Pandemie zermürbt sind, schaffen sie es nicht mehr, die Transaktionen schnell zu berechnen. Setzen Sie das um, indem nach dem Auslesen des Kontostandes und dessen Ausgabe mit **sleep(3)** für einige Sekunden gewartet wird, bevor der aktualisierte Kontostand in die globale Variable geschrieben wird. Da das Sammeln von Werbegeldern zur Zeit sehr anstrengend ist, soll in der Funktion der Finanzierungsgruppe für 8 Sekunden gewartet werden. In der Verpflegungsgruppe soll für 2 Sekunden und in den anderen Gruppen für 3 Sekunden gewartet werden.

→ `weihnachtsmarkt_a2.c`

3 Synchronisation (2 Punkte)

Wenn Sie die Ausgaben Ihres Programms aus Aufgabe 2 betrachten, fällt Ihnen vermutlich auf, dass ein Fehler in der Berechnung der Bilanz aufgetreten ist. Lösen Sie dieses Problem mithilfe der Funktionen **pthread_mutex_init(3)**, **pthread_mutex_lock(3)** und **pthread_mutex_unlock(3)**. Denken Sie daran, am Ende von main() mit **pthread_mutex_destroy(3)** den Mutex wieder zu entfernen.

Da die Arbeitsgruppen ihr Abrechnungproblem jetzt gelöst haben, können Sie den Kontostand viel schneller anpassen und nutzen die gesparte Denkzeit nun als Ruhepause am Ende der Abrechnungsiteration. Verschieben Sie also den Aufruf von sleep direkt vor das Ende des Schleifenrumpfes.

→ `weihnachtsmarkt_a3.c`

4 Bedingungsvariablen (5 Punkte)

Bisher konnte es vorkommen, dass eine Transaktion nicht gedeckt war. Da die Bank aber durch die Ereignisse der letzten Wochen etwas nervös geworden ist, wurde dem ORGA-Team der Dispo-Kredit gestrichen.

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, wenn nicht genügend Geld auf dem Konto vorhanden ist (**pthread_cond_wait(3)**) und erst dann fortfahren, wenn ein Geldeingang auf dem Konto zu verzeichnen ist (**pthread_cond_signal(3)**). Nutzen Sie dazu Bedingungsvariablen (**pthread_cond_init(3)**, **pthread_cond_destroy(3)**) und erweitern Sie Ihr Programm entsprechend.

Hinweise:

- Überlegen Sie zuerst, welche Ihrer Threads auf Deckung ihrer Transaktionen warten müssen und an welchen Stellen den wartenden Threads mitgeteilt werden muss, dass neues Guthaben auf dem Konto vorhanden ist.

→ `weihnachtsmarkt_a4.c`

Tipps zu den Programmieraufgaben:

- Kommentieren Sie Ihren Quellcode ausführlich, so dass wir auch bei Programmierfehlern im Zweifelsfall noch Punkte vergeben können!
- Denken Sie daran, dass viele Systemaufrufe fehlschlagen können! Fangen Sie diese Fehlerfälle ab (die Aufrufe melden dies über bestimmte Rückgabewerte, siehe die jeweiligen man-Pages), geben Sie geeignete Fehlermeldungen aus (z.B. unter Zuhilfenahme von **perror(3)**) und beenden Ihr Programm danach ordnungsgemäß.
- Die Programme sollen dem C11- und POSIX-Standard entsprechen sich mit dem gcc auf aktuellen Linux-Rechnern wie denen im CIP-Pool oder der BSVM übersetzen lassen.
  `gcc -std=c11 -Wall -D_GNU_SOURCE -pthread -o weihnachtsmarkt weihnachtsmarkt_ax.c`
  Weitere (nicht zwingend zu verwendende) nützliche Compilerflags sind: `-Wextra -Wpedantic -Werror -D_POSIX_SOURCE`
- Achten Sie darauf, dass sich der Programmcode ohne Warnungen übersetzen lässt, z.B. durch Nutzung von `-Werror`.