

BETRIEBSSYSTEME - PRAKTIKUM 2

Sommersemester 2022

AUFGABE 1: VORBEREITUNG

- a) Lesen Sie die man-pages zu den POSIX-Funktionen `pthread_mutex_init`, `pthread_mutex_destroy`, `pthread_mutex_lock` und `pthread_mutex_unlock` durch.
- b) Starten Sie das Programm *System Monitor* (oder ein anderes Programm Ihrer Wahl) und beobachten Sie die CPU-Auslastung.
- c) Lesen Sie die man page zum Befehl *time(1)*. Informieren Sie sich über die genaue Bedeutung die drei Ausgaben "real", "user" und "sys" haben (z.B. auf https://www.thomas-krenn.com/de/wiki/Linux_Analyse_der_Ausf%C3%BChrungszeit_mit_time).
- d) Lesen Sie die man page zum Befehl *taskset(1)*. Informieren Sie sich darüber, wie man einen Prozess an eine CPU binden kann.
- e) Laden Sie sich die drei Rahmenprogramme aus dem Moodle-Kurs herunter und schreiben Sie ein Makefile um diese zu bauen.

AUFGABE 2: SYNCHRONISIERUNG

- a) Kompilieren Sie die Datei *inc_threads.c* mit Hilfe der Makefile, die Sie in der Vorbereitung erstellt haben
- b) Lesen Sie den Quellcode und überlegen Sie sich, welchen Wert die Variable *x* am Ende des Programms annimmt. Führen Sie das Programm danach mehrfach aus und prüfen Sie ob Ihre Erwartung zutreffen. Warum ist die Ausgabe jedes mal eine andere?
- c) Erweitern Sie das Programm *inc_threads_solution.c* so, dass der Zugriff auf *x* durch eine *pthread_mutex_t* geschützt ist. Achten Sie darauf, dass nur der kleinstmögliche kritische Abschnitt gelockt wird. Kompilieren Sie Ihre Lösung und stellen Sie sicher dass es den korrekten Wert produziert.
- d) Nutzen Sie das Programm *time(1)* um die Performance von *inc_threads* und *inc_threads_solution* zu vergleichen. Erklären Sie die Unterschiede.
- e) Kopieren Sie das Programm *inc_threads_solution.c* nach *inc_threads_solution2.c*. Versuchen Sie das Programm so zu verbessern, dass durch geeignetes Locking die Performance gesteigert wird. Messen Sie mit *time(1)* ob die Performance besser wird. Welche Auswirkungen hat dies auf die parallele Ausführung Ihres Programms?
- f) Mit dem Programm *taskset(1)* kann ein Programm an eine oder mehrere CPUs gepinnt werden. Nutzen Sie das Programm *taskset* um Ihre drei Lösungen zunächst auf einer CPU auszuführen. Messen Sie die Ausführungszeit mit *time(1)*. Wiederholen Sie das Experiment mit zwei CPUs. Welche Unterschiede können Sie erkennen? Woran könnte dies liegen?

AUFGABE 3: SYNCHRONISATION AUF ZWEI VARIABLEN

- Kompilieren Sie das Programm *inc_two_variables.c* mit Hilfe der Makefile aus der Vorbereitung.
- Führen Sie das Programm aus und beobachten Sie die CPU-Auslastung. Terminiert das Programm? Was könnte das Problem sein?
- Nutzen Sie einen Debugger (z.B. *gdbtui* oder *ddd*, falls notwendig installieren Sie diese mit *sudo apt install gdb ddd*) um Ihre Hypothese zu prüfen. Führen Sie dazu das Programm aus, und unterbrechen Sie den Ablauf des Programms (im Terminal durch die Tasten-Kombination *Strg+C*). Nutzen Sie den Befehl *info threads* um sich den Zustand der Threads anzuschauen. Können Sie das Problem erkennen?
- Beheben Sie das Problem und prüfen Sie die Korrektheit Ihrer Lösung durch Kontrolle der des richtigen Wertes der Ausgabe.

AUFGABE 4: BANK-SIMULATION

Gegeben sei eine stark vereinfachte Bank-Simulation in *banking.c*. Die Bank hat *NUM_KONTEN*, auf denen ein zufälliger Startbetrag ist. Es gibt eine Menge von *THREADS* viele Threads. Diese überweisen jeweils *TRANSFER_PRO_THREADS* oft einen zufälligen Geldbetrag von einem zufälligen Konto auf ein anderes zufälliges Konto. Die relevante Funktion ist die Funktion *ueberweisung*, die parallel ausgeführt wird.

Der Rest des Programms führt die Initialisierung der Konten durch und startet die Threads. Am Ende werden die Kontostände ausgegeben. Die Summe der Kontostände am Anfang und am Ende müssen gleich sein, sonst liegt ein Programmfehler vor.

- Überlegen Sie sich, warum die Funktion *ueberweisung* zu Beginn prüft ob das Quell- und Zielkonto gleich sind. Welches Problem behebt diese Prüfung?
- Starten Sie das Programm. Warum funktioniert das Programm nicht? Welche Art von Problem liegt vor und warum? (Hinweis: Suchen Sie den Fehler nur in der Funktion *ueberweisung*)
- Zeichnen Sie für eine mögliche Fehlersituation den Belegungs-Anforderungs-Graphen.
- Verändern Sie testweise die Parameter *NUM_KONTEN*, *THREADS*, *TRANSFERS_PRO_THREAD*. Setzen Sie z.B. *NUM_KONTEN* auf 1000 und *TRANSFERS_PRO_THREAD* herab auf 100.
Läuft das Programm mit diesem Parametern? Warum?
- Schreiben Sie *ueberweisung* so um, dass die Funktion immer reibungslos funktioniert. Es gibt mindestens zwei Lösungsansätze.
- Überlegen Sie sich welche der vier Deadlock-Bedingungen mit Ihrer Lösung gebrochen wurden. Wie könnten Ansätze aussehen, die ein anderes Kriterium brechen?