Praktische Übungsserie 2

Synchronisation

Aufgabe 1:

In dieser Aufgabe geht es darum, eine endliche Anzahl von Ressourcen zu verwalten. Als Anwendungsbeispiel kannst du dir die Verwaltung einer gewissen Anzahl von *concurrent* Lizenzen bei proprietärer Software denken: Die Anzahl an gleichzeitig ausführbaren Programm-Instanzen ist beschränkt. Wird das Programm ausgeführt, so wird ein Counter dekrementiert. Wird das Programm geschlossen, so wird der Counter wieder inkrementiert. Wenn alle verfügbaren Lizenzen in Gebrauch sind, werden Anfragen um das Programm zu starten zurückgewiesen.
Im Skeleton-File ***`resources.c`*** findest du eine Vorlage für eine derartige Ressourcen-Verwaltung: Die Funktion ***`decrease_count()`*** wird aufgerufen, wenn eine Ressource beansprucht wird. Zum Freigeben einer Ressource wird ***`increase_count()`*** gerufen.
Das zur Verfügung gestellte Skeleton-File führt aber zu einer *Race Condition*.

a) Identifiziere die Variablen und Daten, welche von der *Race Condition* beeinträchtigt werden. Begründe.
b) Identifiziere die Zeilen im Code, welche zur *Race Condition* führen. Begründe.
c) Erstelle zwei Threads, welche die Funktion ***`runTimes()`*** aufrufen, um die Race Condition sichtbar zu machen. Abhängig von den Leistungsparametern deines Rechners solltest du jedes zweite bis dritte Mal eine Abweichung vom Soll-Wert am Ende der Ausführung erhalten. Allenfalls musst du dafür den Wert der Variable ***`times`*** anpassen.
d) Benutze einen *Semaphore* oder *Mutex Lock*, um die *Race Condition* zu beheben. Du darfst dafür die Funktionen ***`decrease_count()`*** und ***`increase_count()`*** so anpassen, dass der aufrufende Prozess blockiert wird.

Benutze das Skeleton *`resources.c`*
Beantworte Fragen *a)* und *b)* in einem Textfile *Answers.txt*

Befehl zum Kompilieren: gcc -Wall -std=c99 -pthread resources.c -o resources

Benutze C99 um die Übungen zu lösen

Abgabefrist: 26. März 2019, 10h00