Lisa Obermaier
Simon Thum

05.06.2019

4. Zyklische, kommunizierende Tasks

a) *Erstellung der Tasks*

In der *main*-Funktion werden zwei Threads erstellt: *threadSetSemaphore* und *threadWaitForSemaphore*. Sie werden beide mit *pthread_create* gestartet und am Ende der Funktion über *pthread_join* wieder beendet, sobald beide Tasks ihre Schleifen abgearbeitet haben.

threadSetSemaphore erhält zu Beginn die Priorität 50. Anschließend startet eine Schleife, die 1000-mal durchlaufen wird, um eine ausreichend lange Laufzeit für den Kernel Tracer zu erreichen. Mit jedem Schleifendurchlauf wird die aktuelle Zeit gemessen und in der Variable *deadline* die absolute Zielzeit in vier Millisekunden gespeichert. Anschließend wird versucht mit *waste_msec* annähernd zwei Millisekunden zu arbeiten und mit *clock_nanosleep* die verbleibenden Millisekunden zu schlafen. Jeden dritten Schleifendurchlauf wird die Semaphore für den anderen Thread freigegeben.

threadWaitForSemaphore erhält zu Beginn die Priorität 100, wird also bei der Abarbeitung bevorzugt vor dem ersten Thread behandelt. Anschließend durchläuft auch dieser Thread 1000-mal eine Schleife, in der auf die Freigabe der Semaphore getestet wird. Wenn die Semaphore freigegeben wurde, arbeitet der Thread drei Millisekunden mit der *waste_msecs*-Funktion.

b) *Semaphore*

Die Semaphore wird initialisiert mit *sem_init* in der *main*-Methode, nachdem die Semaphore selbst als global deklariert wurde und aus jeder Stelle des Programms darauf zugegriffen werden kann. *threadSetSemaphore* gibt die Semaphore mit *sem_post* frei, *threadWaitForSemaphore* wartet mit *sem_wait* auf die Freigabe.

c) *Kernel Tracer*

Die Ausgabe des *Kernel Tracers* zeigt gut, dass der erste Thread, *threadSetSemaphore*, drei Millisekunden *ready* ist, in denen der zweite Thread, *threadWaitForSeaphore*, die Möglichkeit zu arbeiten hat. Anschließend arbeitet der erste Thread zwei Millisekunden, anschließend schläft er zwei Millisekunden und solange die Semaphore nicht alle drei Durchläufe freigegeben wird, wechseln sich Arbeiten und Schlafen des ersten Threads ab, bis der zweite Thread wieder drei Millisekunden arbeiten kann und der erste Thread solange *ready* ist.

Lisa Obermaier
Simon Thum

05.06.2019