Prof. Dr. Jens-Peter Akelbein
Olga Dedi

Threads/Scheduling

1 Sichtbares Scheduling

Ziel ist es, dass Sie das Verhalten des Schedulings und so den Wechsel zwischen Threads so sichtbar wie möglich machen und Einflüsse darauf erkennen.

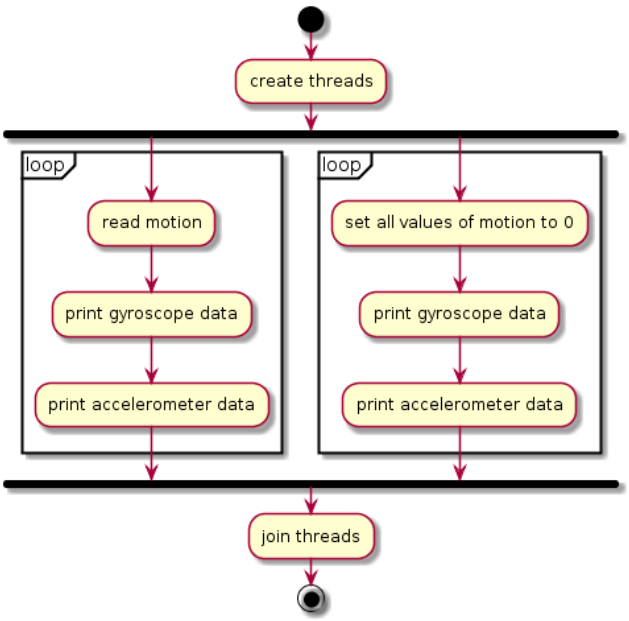

Figure 1: Aktivitätsdiagramm der zu erstellenden Threads

1. Schreiben die ein Programm, welches das Aktivitätsdiagramm in Abbildung 1 implementiert. Erzeugen Sie dafür zwei Threads im Hauptprogramm. Erzeugen Sie eine *Motion_t* Struktur, die beiden Threads übergeben wird. Thread 1 soll mithilfe der Klasse SensorTag aus Termin 1 die Struktur füllen und anschließend die Werte ausgeben. Thread 2 soll alle Werte auf 0 setzen und diese ebenfalls ausgeben.
   Tip: Die pthread Library fügen Sie in Code::Blocks unter Project→Build Options→ Linker Settings hinzu.

2. Lassen Sie Ihr Programm im Terminal und nicht aus der IDE laufen. Was fällt Ihnen auf?

3. Rufen Sie Ihr Programm mit strace auf. Versuchen Sie, im Trace von strace die Systemaufrufe zu identifizieren. Welcher System Call führt die Ausgaben von *printf* bzw. *cout* durch?

   ```
   strace -f ./executable 1> outfile 2> tracefile
   ```

4. Verändern Sie die Anzahl der Schleifendurchläufe auf 10, 1000 und 100000 und lassen Sie Ihr Programm laufen. Welchen Puffer gibt es?

5. Versuchen Sie, die Threads mithilfe von *usleep* bzw. *nanosleep* zu synchronisieren, so dass sich die beiden Threads wie in Abbildung 2 dargestellt abwechseln. Nutzen Sie *gettimeofday*, um heraus zu finden, wie lange eine Iteration dauert und wie lange die jeweiligen Threads schlafen müssen, um sich nach jedem Schritt abzuwechseln.

6. Verändern Sie die Anzahl der durch Ihr Programm verwendeten Kerne mit dem CLI-Tool taskset. Was verändert sich in der strace-Ausgabe?

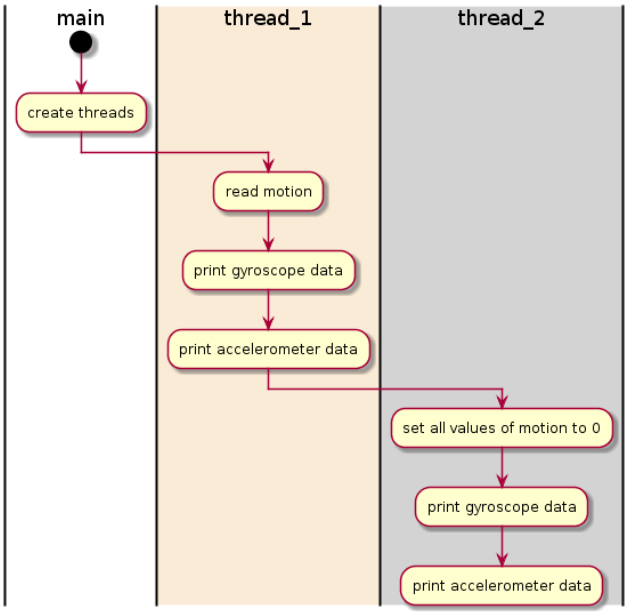

Figure 2: Synchronisation der zu erstellenden Threads

2 Synchronisierter Wechsel zwischen Threads

Erstellen Sie eine Kopie Ihres Projekts und entfernen Sie die Zeitmessung und usleep-Aufrufe. Ziel der Aufgabe ist es, eine möglichst gute Synchronisation zwischen den beiden Threads zu erreichen, deren Scheduling mit Realtime-Priorität nach der FIFO-Policy erfolgt. Dazu sollen diese wie bereits mit usleep nachgestellt jeweils abwechselnd eine Iteration Ihrer Schleife durchführen. Testen Sie Ihr Programm nach jedem Zwischenschritt.

1. Lassen Sie Ihr Programm ohne die usleep-Aufrufe laufen.

2. Ändern Sie die Attribute des zu erzeugenden Threads. Kapseln Sie die Initialisierung eines Thread-Attributs in einer eigenen Funktion. Ändern Sie die Scheduling-Policy und –priorität. Wie ändert sich der Ablauf? Wie würden sich andere Scheduling-Policies auswirken?

3. Konfigurieren Sie die durch Ihr Programm verwendeten CPU-Ressourcen so, so dass ein synchroner Ablauf möglich wird.

4. Geben Sie die CPU ab, indem Sie nanosleep verwenden. Variieren Sie die Wartezeiten in der Workerthread-Funktion im Bereich von einer Mikrosekunde zu bis mehreren Millisekunden, bis Sie möglichst stabil zwischen den beiden Workerthreads umschalten. Lenken Sie die Ausgabe in eine Datei um, um Einflüsse durch die Remote-Anzeige im Terminal auszuschließen. Wie verhält sich Ihr Programm bei kurzen Wartezeiten und langen Programmläufen?

5. Verwenden Sie eine bessere Möglichkeit, wie Ihr Programm die CPU abgibt. Schauen Sie sich dafür die Funktion *sched_yield* an. Vergleichen Sie den Effekt bei längeren Laufzeiten.