| h_da | Embedded SW-Praktikum | |
|---|---|---|
| Termin 2 | Threads/Scheduling | Prof. Akelbein |

## Teil 1: sichtbares Scheduling

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

1. Beantworten Sie alle Fragen in diesem Text anhand Ihrer Beobachtungen. Nehmen Sie das Beispielprojekt thread_cout.
2. Zur besseren Sichtbarkeit ersetzen Sie die Ausgabe von Zahlen und Klammern einfach durch zwei unterschiedliche Buchstaben für die beiden Threads. Ersetzen Sie cout durch printf() und cerr durch fprintf(stderr, ...).
3. Lassen Sie das Programm in einem eigenen Terminal und nicht aus der IDE laufen. Hier können Sie es mit ohne und mit strace aufrufen. Versuchen Sie im Trace von strace die Systemaufrufe zu identifizieren. Welcher System Call führt die Ausgaben von printf() durch?
4. Entfernen Sie die sleep() Aufrufe in dem Programm, ändern Sie die Anzahl Schleifen-durchläufe auf 10, 1000 und 100000 und lassen Sie Ihr Programm laufen. Welchen Puffer gibt es?
5. Wie wirkt sich der Aufruf von fflush() aus, wenn Sie dies nach jeder Ausgabe durch printf() aufrufen? Was passiert, wenn Zeichen immer abwechselnd ausgegeben werden? Was sehen Sie an dieser Stelle mit strace?
6. Verändern Sie die Anzahl der durch ihr Programm verwendeten Kerne mit dem CLI-Tool taskset. Was verändert sich in der strace-Ausgabe?

### Tipps:

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

So tracen Sie alle Threads und Kindprozesse (Option –f) und lenken die Ausgaben über stdout (1) und stderr (2) in Dateien um. Öffnen Sie diese Dateien in Code::Blocks, um jeweils die aktuelle Ausgabe und Trace nach jedem Programmlauf im Editor zu sehen.

Zeilenumbrüche:

Führen Sie eine globale (volatile) Variable ein, die Sie in beiden Threads bei jeder Ausgabe inkrementieren. Mit dem %-Operator können Sie in jeder x-ten Zeile ein \n ausgeben z.B.

mit der globalen Variable column und if ( ++column % 100 ) printf(´´\n´´);
nach jedem 100sten Zeichen.

**Teil 2: Synchronisierter Wechsel zwischen Threads**

Erstellen Sie ein neues Projekt. Nehmen Sie als Startquelle das Ergebnis des ersten Teils. Ziel der Aufgabe ist es, eine möglichst gute Synchronisation zwischen zwei Worker Threads zu erreichen, deren Scheduling mit Realtime-Priorität nach der FIFO-Policy erfolgt. Dazu sollen diese jeweils abwechselnd eine Zeile mit ihrem Zeichen ausgeben. Testen Sie Ihr Programm nach jedem Zwischenschritt.

1. Entfernen Sie die Bildschirmausgaben in der main(). Nun erzeugen Sie aus main() zwei Workerthreads, die mit der gleichen Funktion starten. Als Parameter übergeben Sie statt einem Zeiger auf int einen Zeiger auf eine Zeichenkette, die das Zeichen enthält, was der Thread als Kennzeichen ausgeben soll. Jeder der beiden Threads gibt ein anderes Zeichen aus.
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 (>> 1 Sekunde)?
5. Verwenden Sie eine bessere Möglichkeit, wie Ihr Programm die CPU abgibt. Vergleichen Sie den Effekt bei längeren Laufzeiten.

**Tipps:**

strace:

Lassen Sie strace laufen, um die Störungen im Ablauf auf Systemcall-Ebene zu sehen.

Beispielprogramm:

Sie finden in der Man-Page Beschreibungen für pthread-Funktionen. In der Beschreibung zur Attributinitalisierung finden Sie ein Beispielprogramm.

nanosleep()

Mit dem System Call nanosleep() können Sie Threads kurzfristig schlafen lassen. Die man-Page enthält die Angaben zu den Parametern.

htop

htop zeigt Ihnen die aktuellen Prozesse und Threads an. Wenn Sie nach Ihrem Prozess suchen, können Sie sehen, welche Threads dieser erzeugt hat. Öffnen Sie htop hierzu in einer xterm-Konsole.

Tipps für Code::Blocks bei Erstellung neuer Projekte

pthread-Library mit dazulinken
Dazu gehen Sie in Project->Build Options ->Linker Settings und fügen mittels Add die Library pthread hinzu.