Blatt 06

Abgabe bis: 01. Dezember 2011

9. Cachedetect (10P)

Dieses Mal geht es um den Speicher und insbesondere um die Caches. Ziel ist es, herauszufinden, welche Zugriffszeiten Arbeitsspeicher und Caches haben und wieviele Caches von welcher Groesse vorhanden sind.

Dazu sollt ihr ein Programm schreiben, das in mehreren Durchgaengen ein immer groesseres Array anlegt und viele Lesezugriffe darin erzeugt. Die durchschnittliche Dauer eines Zugriffs bei einer bestimmten Arraygroesse soll am Ende jedes Durchgangs ausgegeben werden. Es wird sich zeigen, dass die Zugriffsdauern von der Groesse des Arrays abhaengen, allerdings nicht linear oder exponentiell. Vielmehr werden sich Treppen bilden ... in der Groesse der verbauten Caches.

Bei den Lesezugriffen ist es wichtig, dass Optimierungen, wie Pipelining oder Vorausahnen von zukuenftigen Aktionen, ausgehebelt werden. Fuellt dazu den Speicher mit einem Array von Structs, die jeweils nur einen next-Zeiger auf ein gleiches Struct beinhalten. Verkettet alles Structs mit diesem Zeiger untereinandern in Form eines Rings. Die Reihenfolge der Verkettung muss dabei zufaellig sein. (Hilfestellung: Legt ein Hilfsarray von Indices an, mischt diese mit dem Fisher-Yates Shuffle-Algorithmus, verkettet dann die Structs entsprechend der Reihenfolge des gemischten Hilfsarray.) Nun koennt ihr vom ersten Struct aus dem next-Zeiger beliebig oft folgen, unabhaengig davon aus wievielen Elementen der Ring besteht. Die Zugriffe werden gleichmaessig auf das Array verteilt sein und die Position jedes weiteren Zugriffs ist vom aktuellen Element abhaengig. Achtet darauf, dass ihr beim Kompilieren nicht optimiert, denn sonst kann euch ein zu intelligenter Optimierer den ganzen Spass verderben.

Ein sinnvoller Bereich fuer die Arraygroesse ist 1 KB bis 32 (oder gar 64) MB. Um Messfehler zu reduzieren, ist es angebracht, dem next-Zeiger sehr oft nachzulaufen. Man sollte zumindest 16 mal durch den Ring gelaufen sein, bei einer kleinen Anzahl von Elementen aber deutlich oefter.

Bei euren momentanen C-Kenntnissen ist noch aller Speicher auf dem Stack. Die Stackgroesse ist vom Betriebssystem begrenzt, oft auf 8192 KB. Diesen Wert solltet ihr fuer diese Aufgabe deutlich erhoehen. Mit folgendem Shell-Befehl bekommt ihr angenehme 64 MB: `ulimit -s 66000`.

Zum Zeitmessen kann die Funktion *times()* verwendet werden. Folgendes Dummy-Programm zeigt eine beispielhafte Verwendung:

```
#include <stdio.h>
#include <sys/times.h>
#include <limits.h>
#include <unistd.h>

int
main(void)
{
        /* CLK_TCK: ticks per second -- is system dependent */
#ifndef CLK_TCK
        int CLK_TCK = sysconf(_SC_CLK_TCK);
#endif
        clock_t ticks;
        double nanosecs;
        struct tms timebuf1, timebuf2;

        times(&timebuf1);  /* start */
        /* do what you want to measure */
        times(&timebuf2);  /* stop */

        /* get number of ticks */
        ticks = timebuf2.tms_utime - timebuf1.tms_utime;

        /* convert to human readable unit: nano seconds */
        nanosecs = (double)ticks / CLK_TCK * 1000000000;
        printf("duration: %f ns\n", nanosecs);

        return 0;
}
```

Um das Ergebnis anschaulich darzustellen, koennt ihr *gnuplot* verwenden. Gebt dazu pro Messwert (= Durchgang) eine Zeile aus: x-Wert (= Arraygroesse) und y-Wert (= Dauer) durch Tab getrennt. Wenn ihr eure Ausgabe in eine Datei umleitet, kann gnuplot daraus ein Diagramm erzeugen:

```
$ ./cachedetect >data
$ echo 'set logscale x; plot "data" with lines' | gnuplot -persist
$
```

Gnuplot zeigt das Diagramm standardmaessig in einem grafischen Fenster an. Wenn ihr auf einen entfernten Rechner (z.B. theseus, der sehr grosse Caches hat) arbeitet, dann muesst ihr *ssh* mit den Optionen `-X -Y` verwenden. Alternativ koennt ihr mit dem gnuplot-Befehl '`set terminal png;`' auch Bilder erzeugen.

Startet das Programm auf unterschiedlichen Rechnern und vergleicht die Ergebnisse.

Zum Einreichen eurer Loesung:

```
submit ss1 9 team [notes] cachedetect.c
```

Fragen zur Selbstkontrolle

- Wieviel Speicher belegt ein Struct?
- Gebe ein Beispiel zur sinnvollen Verwendung einer Union.

- Weshalb verpackt man eine Union oft in ein Struct?
- Was versteht man unter einem anonymen Struct?
- Wie Initialisiert man ein neu definiertes Struct?
- Demonstriere die Verwendung des `->` Operators.
- Was darf man aus Funktionen zurueckgeben?
- Erklaere typedef.