LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Grundlagenpraktikum: Rechnerarchitektur

Arbeitsblatt 8
13.06.2022 - 19.06.2022

T8.1 Valgrind

Früher oder später schleicht sich in so gut wie jedes C Programm ein Fehler bezüglich der Speicherallokation oder -freigabe, oder bezüglich Speicherzugriffen ein. Gerade diese sind allerdings häufig schwer zu debuggen, da der Fehler oftmals nicht im direkten lokalen und/oder temporalen Kontext der eigentlichen Ursache auftritt. Wir möchten uns anhand des Programms `outerprod` nun ansehen, wie uns das Tool Valgrind helfen kann, die Ursache dieser Fehler dennoch aufzuspüren und zu beheben.

Material: `https://gra.caps.in.tum.de/m/outerprod.tar`

1. Besprechen Sie mit Ihrem Tutor zunächst den Aufbau und die grobe Funktionsweise des Programms.
2. Kompilieren Sie das Programm mit `make` und führen Sie es aus. Verhält es sich wie erwartet?
3. Führen Sie das Programm zum Debuggen nun mit GDB aus (`gdb ./outerprod; r`). An welcher Stelle der Datei `outerprod.c` stürzt das Programm ab?
4. Lassen Sie sich mit dem Befehl `p mat` den Pointer auf die Ausgabematrix anzeigen. Können Sie allein hieraus erkennen, warum das Programm abstürzt?
5. Führen Sie das Programm nun mit Valgrind aus: `valgrind ./outerprod`. Wie interpretieren Sie die Ausgabe? Was ist die *unmittelbare* Ursache für den Absturz?
6. Wie groß ist der Speicherbereich, auf den zum Zeitpunkt des Programmabsturz zugegriffen wird? Wie groß *sollte* er sein?
7. Wie erklären Sie sich diese Diskrepanz?
8. Informieren Sie sich im GCC Manual[2] über die Integer Overflow Built-ins und überlegen Sie, wie Sie diese nutzen können, um dieses und ähnliche Probleme allgemein zu verhindern.

S8.1 Leistungsoptimierung

Material: `https://gra.caps.in.tum.de/m/benchmark.tar`

[2] `https://gcc.gnu.org/onlinedocs/gcc/Integer-Overflow-Builtins.html`

1. Kompilieren Sie das Programm `matr` mit `make` und führen Sie es mit `time ./matr` aus. Welche benötigte Laufzeit gibt das Programm an? Wie groß ist der Unterschied zwischen der real benötigten Zeit und der angezeigten Zeit?

2. Um den Grund für den Unterschied herauszufinden, verwenden wir nun den *Profiler* `perf`. Dieser ist auf der `lxhalle` bereits installiert.[3] Führen Sie das Programm nun mit `perf record`[4] aus:

   ```
   perf record ./matr
   ```

   Und lassen Sie sich nach der Ausführung das Resultat anzeigen:

   ```
   perf report
   ```

   In welchen Funktionen wird am meisten Zeit verbracht? Entfernen Sie den (unnötigen) Aufruf in diese Funktion.

3. Kompilieren Sie das Programm erneut und führen Sie es erneut mit `perf record` aus. Lassen Sie sich wieder anzeigen, wo das Programm die meiste Zeit verbringt. Wie lässt sich das Problem beheben?

   Hinweis: Es empfiehlt sich, die Zahl der Iterationen beim Programmaufruf wie folgt zu erhöhen: `./matr -i 50`

4. Nun kann `perf` nicht nur die benötigte Zeit anzeigen, sondern auch viele weitere Informationen, welche aus sog. *Performance Countern* des Prozessors ausgelesen werden können. Diese zählen im Hintergrund verschiedene Events mit, welche dabei helfen können, Performanzprobleme zu analysieren. Führen Sie `perf list` aus, um eine Übersicht über die Events, die von dem jeweiligen System (Prozessor, Betriebssystem) unterstützt werden, zu erhalten.

5. Im Folgenden betrachten wir das Event `cache-misses`, welches die Anzahl der Speicherzugriffe zählt, wo das Resultat nicht im schnellen Cache im Prozessor liegt, sondern erst aus dem verhältnismäßig langsamen Hauptspeicher geholt werden muss.

   Führen Sie das Programm wie folgt aus, um anstelle der Berechnungszeit die Zahl der Cache-Misses zu messen:

   ```
   perf record -e cache-misses ./matr -i 50
   ```

   ```
   perf report
   ```

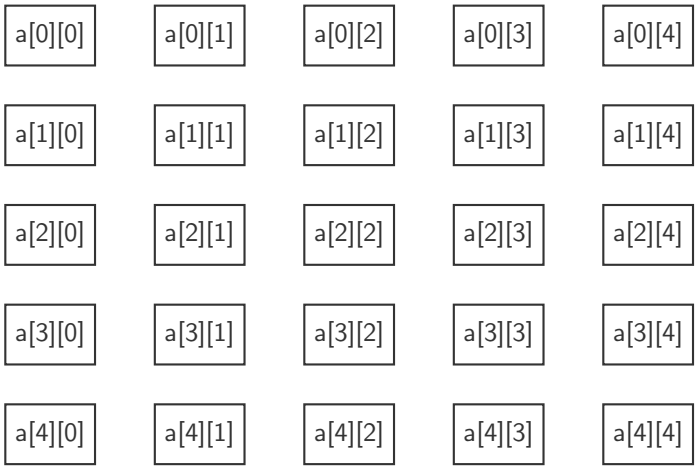
[3] Wir empfehlen Ihnen, soweit möglich, ein eigenes System für diese Aufgabe zu verwenden, da die `lxhalle` eine sehr restriktive Konfiguration hat.

[4] Dieser Befehl führt Ihr Programm aus, während `perf` im Hintergrund in regelmäßigen Abständen aufzeichnet, an welcher Stelle sich Ihr Programm befindet. Diese Information wird in der Datei `perf.data` gespeichert.

Genauere Informationen können Sie erhalten, indem Sie eine Funktion auswählen (Navigation über Pfeil-Tasten, Auswahl mit *Enter*) und in dem folgenden Menü den Punkt *Annotate* wählen. Sie können die Ansicht mit `q` oder `Ctrl-C` verlassen.

An welchen Stellen verursacht das Programm viele Cache-Misses?

6. Überlegen Sie sich, wie die Matrix im Speicher abgelegt ist und in welcher Reihenfolge darauf zugegriffen wird. Zeichnen Sie beide Reihenfolgen in folgende Grafik ein.



Durch welche kleine Veränderung lässt sich die Cache-Nutzung *deutlich* verbessern und damit die Performanz signifikant steigern?

P8.1 Hamming-Distanz [4 Pkt.]

Die Pinguine der Gattung ERA (kurz: ERA-P) verfügen über Augen, die lediglich Helligkeitswerte wahrnehmen[5] und diese pro wahrgenommen Bildpunkt digital im Wertebereich von 0–255 an das Gehirn weitergeben; eine Wahrnehmung ist dabei eine Liste bekannter Länge von Bildpunktdaten. Im Gehirn eines ERA-Pinguin sind Bilder von vergangenen Wahrnehmungen gespeichert. Ein Pinguin orientiert sich, in dem die aktuelle Wahrnehmung mit vormaligen Wahrnehmungen verglichen wird.

Da aber weder die Erinnerung der ERA-Pinguine noch die Übertragung der Wahrnehmung von den Augen an das Gehirn Error-Correcting-Codes benutzt, gehen sporadisch Daten verloren: Bei einigen Wahrnehmungen in der Erinnerung sind einzelne

[5] Die Pinguine entstammen der Schwarz-Weiß-ERA.

Bildpunkte nicht korrekt und bei der Übertragung der Daten von den Augen kommen manche Bildpunktdaten am Ende nur als zufälliges Rauschen an.

Deswegen realisieren die ERA-Pinguine den Bildvergleich, in dem die Anzahl der abweichenden Bildpunkte gezählt wird (*Hamming Distance*); die Erinnerung mit der geringsten Abweichung wird dann zur Orientierung genutzt. Insbesondere im Wasser ist eine extrem schnelle Orientierung überlebenswichtig, weshalb der Vergleich zwischen Wahrnehmung und Erinnerung so schnell wie möglich geschehen muss.

Aufgabe: Optimieren Sie folgende Funktion, welche die Anzahl der unterschiedlichen Elemente der Arrays `a` und `b` der gleichen Länge `n` berechnet, um die Überlebensfähigkeit der ERA-Pinguine zu erhöhen!

```
size_t hamming_dist(size_t n, const char a[n], const char b[n]) {
  size_t res = 0;
  for (size_t i = 0; i < n; i++)
    res += a[i] != b[i];
  return res;
}
```

X8.1 ToUpper: MemeAssembly-Edition [2 Pkt. Bonus]

In dieser Einheit betrachten wir erneut die exzellente Funktion `toupper`, welche in einem String sämtliche unexzellente Klein-Buchstaben durch die entsprechenden exzellenten Groß-Buchstaben ersetzt. Diese Aufgabe wird Ihnen eine Einführung in die exzellente Programmiersprache MemeAssembly geben.

1. Verwenden Sie die Dokumentation[6], um für die folgenden Probleme geeignete Commands zu finden:
   - Laden der Zahl 42 in das Register `rax`
   - `rcx` auf den Stack pushen
   - Sprung, falls `rax` = 0
   - Ein Byte von der Speicheradresse in `rdi` in das Register `al` laden

     Hinweis: Der Register-Suffix `do you know de wey` führt zu einem Speicherzugriff (äquivalent zu `[Register]`)
   - Vergleich zwischen `al` und 97, mit dem Sprung zu einem jeweiligen Marker – je nachdem, welcher Wert größer ist
   - Berechnen von 2^3 in `rax`
   - Programmabsturz durch einen Segmentation Fault
   - Durch Zufall während des Kompilierens die Hälfte aller Codezeilen löschen

[6] `https://kammt.github.io/MemeAssembly`

2. Beginnen Sie mit dem groben Schleifenkonstrukt: Laden Sie einen Buchstaben aus dem Speicher und springen Sie aus der Schleife heraus, falls es sich um ein `NULL`-Byte handelt. Vergessen Sie dabei nicht, den Pointer zu inkrementieren.

3. Für den Zweck des Vergleichs bietet sich der Command `who would win?` an. Überprüfen Sie nun, ob der Buchstabe ein (unexzellenter) Kleinbuchstabe ist. Beachten Sie hierbei, dass Sie eventuell den (potentiell unexzellenten) Buchstaben in ein anderes Register verschieben müssen, um einen neuen Sprung-Marker erstellen zu können.

4. Verwandeln Sie nun den Buchstaben in einen exzellenten Großbuchstaben und schreiben Sie ihn in den Speicher zurück.

   Erinnerung: Der Suffix `do you know de wey` interpretiert einen Register-Parameter als eine Speicheradresse

5. Sie werden nun feststellen, dass Ihr Code zwar (hoffentlich) funktioniert, jedoch noch zu unexzellent ist, um einen Score zu erhalten. Werden Sie kreativ - wie könnte man den Code so weit verlangsamen, ohne dabei einen Timeout auszulösen?

Q8.1 Quiz [4 Pkt.] (siehe Praktikumswebsite)

Q8.2 Praktikumsordnung [2 Pkt.] (siehe Praktikumswebsite)