

Übungsblatt 2 zur Vorlesung Parallele Programmierung

Abgabe: 23.10.2021, 23:59

Jun.-Prof. Dr. Michael Kuhn (michael.kuhn@ovgu.de)

Michael Blesel (michael.blesel@ovgu.de)

Parallel Computing and I/O • Institut für Intelligente Kooperierende Systeme

Fakultät für Informatik • Otto-von-Guericke-Universität Magdeburg

<https://parcio.ovgu.de>

Um später effizient programmieren zu können, wollen wir uns ein wenig mit der Fehlersuche beschäftigen. Hierzu schauen wir uns den GNU Debugger GDB und den Speicherprüfer memcheck aus dem Valgrind-Projekt an. Diese können sowohl für serielle als auch für parallele Programme genutzt werden. In C ist die Speicherallokation sehr fehlerträchtig, Valgrind ist hierbei sehr hilfreich, um typische Fehler aufzuspüren.

1. Debugging einer einfachen Anwendung (150 Punkte)

Im Verzeichnis `simple` ist ein primitives Programm enthalten, welches mit `make` kompiliert werden kann. Dieses Programm dient dazu, dass Sie sich ein wenig mit GDB und Valgrind beschäftigen. Es enthält lediglich vier Funktionen, welche jeweils einen Zeiger auf eine Zahl oder ein Array mit einer Zahl enthalten und gibt diese dann in der `main`-Funktion aus. Leider enthält dieses Programm diverse Fehler.

- Führen Sie folgende kleinere Tests durch, um GDB kennen zu lernen. Dokumentieren Sie die genutzten Eingabebefehle und die Ausgabe in einer Textdatei. Ein kurzes Tutorial für GDB finden Sie beispielsweise unter <https://www.cs.cmu.edu/~gilpin/tutorial/>. Die wichtigsten Befehle sind außerdem in <https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf> zusammengefasst.
 - Starten Sie das Programm. Dafür übergeben Sie am besten direkt den Programmnamen an GDB (`gdb ./simple`).
 - Platzieren Sie einen Breakpoint auf der Funktion `mistake1`, starten Sie das Programm, geben Sie den Wert von `buf` und `buf[2]` aus. Gehen Sie zur nächsten Zeile und geben Sie beide Werte wieder aus. Von welchem Typ ist `buf`?
 - Platzieren Sie einen Breakpoint in der Funktion `mistake2`, setzen Sie den Programm-
lauf fort, welchen Typ hat `buf`?
 - Setzen Sie den Programm-
lauf fort, welcher Text wird ausgegeben? Lassen Sie sich den Code um diese Stelle herum ausgeben. Welche Frames sind auf dem Stack? Wechseln Sie zu Frame 1. Geben Sie den Inhalt von `p` aus.
 - Rufen Sie innerhalb von GDB die Funktion `mistake4` auf (schauen Sie nach, wie man in GDB Funktionen direkt aufrufen kann).
- Modifizieren Sie das Programm zunächst so, dass es nicht mehr abstürzt. Versuchen Sie die Modifikationen möglichst gering zu halten. Verwenden Sie zunächst GDB, um die Fehlerstellen aufzuspüren. Die Ausgabe soll wie folgt aussehen:

```
1: _1
2: _2
3: _3
4: _4
```

- Nun läuft das Programm, leider enthält es jedoch noch weitere Speicherfehler, die je nach Umgebung (mehr oder weniger zufällig) auftreten können. Modifizieren Sie das Programm unter Zuhilfenahme von Valgrinds memcheck so, dass jede Methode Speicher korrekt reserviert und dass am Ende der Speicher korrekt freigegeben wird. Rufen Sie dafür das Programm mit `valgrind ./simple` auf.

Hinweis: Den Speicher einfach mit `static` zu allokieren ist *nicht* erlaubt. Verzichten Sie darüber hinaus auf globale Arrays und Variablen. Eine kurze Anleitung für Valgrind finden Sie beispielsweise unter <https://www.valgrind.org/docs/manual/quick-start.html>.

Dokumentieren Sie die Fehler, die zu den Abstürzen und Speicherfehlern führen. Notieren Sie hierfür für jeden vorhandenen Fehler die Code-Zeilen, welche fehlerhaft sind und den genauen Grund der Ursache (z. B. Speicher mehrfach freigegeben).

2. Debugging einer komplexen Anwendung (120 Punkte)

Im Verzeichnis `pde` finden Sie ein numerisches Programm zum Lösen von Differentialgleichungen. Grundsätzlich ist das Programm vom Ablauf korrekt, jedoch haben sich durch Unachtsamkeit einige Flüchtigkeitsfehler u. a. bei der Speichernutzung eingeschlichen. Um das Programm zu korrigieren, müssen Sie nicht genau verstehen was berechnet wird, u. U. sollten Sie jedoch mit einem Debugger die Aufrufe ein wenig verfolgen. Korrigieren Sie alle Fehler im Programm. Modifizieren Sie hierbei den Code so wenig wie nötig.

Das Programm sollte dabei wie folgt aufgerufen werden: `./partdiff-seq 1 1 100 1 2 5`

3. Batch Scheduling (60 Punkte)

Beantworten Sie die folgenden Fragen und notieren sie gegebenenfalls die Befehlsausgaben:

1. Was bedeuten die Begriffe Job Scheduler bzw. Batch Scheduling?
2. Welche Aufgaben hat ein Job Scheduler?
3. Nennen Sie drei Beispiele für Job Scheduler.
4. Welcher Job Scheduler wird auf dem Cluster verwendet?
5. Machen Sie sich mit der Manpage von `sbatch` vertraut. Beschreiben Sie die Funktionsweise des Kommandos.
6. Wie lassen sich die aktuellen Jobs und deren Status anzeigen?
7. Gibt es eine Möglichkeit, einen bereits abgeschickten Job zu löschen (bevor oder während er läuft)? Wenn ja, wie?

8. Können auf dem Cluster mehrere Nutzer bzw. Jobs gleichzeitig denselben Knoten nutzen?
9. Finden Sie heraus wie Sie sich den detaillierten Status eines Jobs ausgeben lassen können.
10. Wie können Sie die Priorität Ihrer Jobs nachschauen? Können Sie diese verändern oder beeinflussen?
11. Welche Scheduling-Verfahren sind bei dem auf dem Cluster benutzten System möglich? Welches wird benutzt? Erläutern Sie jedes der Verfahren kurz.
12. Welche unterschiedlichen Partitionen sind auf dem Cluster eingerichtet? Wie kann die zu benutzende Partition geändert werden?
13. Finden Sie heraus, wie Sie einen einzelnen Knoten allokatieren können. Allokieren Sie einen konkreten Knoten (z. B. ant13), bauen Sie eine Verbindung zu diesem Knoten auf und führen sie hostname darauf aus.
14. Wie hoch ist das Timeout auf dem Cluster, bis alle Knoten wieder freigegeben werden?

Abgabe

Als Abgabe erwarten wir ein gemäß den Vorgaben benanntes komprimiertes Archiv (MustermannMusterfrau.tar.gz), das ein gemäß den Vorgaben benanntes Verzeichnis (MustermannMusterfrau) mit folgendem Inhalt enthält:

1. Eine Datei gruppe.txt mit den Gruppenmitgliedern (eines je Zeile) im folgenden Format:

```
Erika Musterfrau <erika.musterfrau@example.com>
Max Mustermann <max.mustermann@example.com>
```
2. Eine Textdatei mit den Ein-/Ausgaben von GDB mit dem Namen simple-gdb.txt, eine Textdatei namens simple-error.txt mit Fehlerbeschreibungen (Ursache, Code-Zeilen) und der überarbeitete Code im Verzeichnis simple (Aufgabe 1)
3. Eine Textdatei pde-error.txt mit Fehlerbeschreibungen (Ursachen, Code-Zeilen) und der überarbeitete Code im Verzeichnis pde (Aufgabe 2)
4. Eine Textdatei antworten.txt mit Ihren Antworten (Aufgabe 3)

Hinweis: Am einfachsten ist es mit `make clean` alle Binärdaten zu entfernen, das vorhandene Verzeichnis umzubenennen und alles wieder in ein Archiv zu packen.

Laden Sie das Archiv unter einer der folgenden URLs hoch:

1. Dienstag 7–9 Uhr (Julian Benda): <https://cloud.ovgu.de/s/o6j7jK4mxLGxA9s>
2. Mittwoch 15–17 Uhr (Michael Blesel): <https://cloud.ovgu.de/s/x3fZCakSR2YXRwe>
3. Mittwoch 15–17 Uhr (Johannes Wünsche): <https://cloud.ovgu.de/s/ostkHwWqiRXpc6N>