Universität Hamburg, Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen
Prof. Dr. T. Ludwig, Dr. Jannek Squar

Übungsblatt 02 zur Vorlesung
Hochleistungsrechnen
im WiSe 2026/2027
Abgabe: 24.10.2026, 23:59

1 Debugging von C-Programmen (300 Punkte)

Um später effizient programmieren zu können, wollen wir uns ein wenig mit der Fehlersuche in (parallelen) Programmen beschäftigen. Hierzu schauen wir uns den GNU Debugger `gdb` und den Speicherprüfer `memcheck` aus der `valgrind`-Tool-Suite an. Diese können sowohl für sequentielle als auch für parallele Programme genutzt werden. In C ist die Speicherallokation sehr fehlerträchtig, `valgrind` hilft hierbei typische Fehler aufzuspüren. Eine Liste von Links zu diesen beiden Programmen und wie diese auf dem Cluster verwendet werden können finden Sie auf unserer Webseite:

`https://wr.informatik.uni-hamburg.de/teaching/ressourcen/debugging`

Auf der Materialienseite befindet sich ein Archiv, in dem Zusatzmaterialien für die folgenden Aufgaben enthalten sind. Entpacken Sie dieses in Ihrem Home-Verzeichnis.

1.1 Debugging mit gdb (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 fünf 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 von `gdb` im PDF:
  - Starten Sie das Programm. Dafür können Sie den Programmnamen direkt an GDB übergeben (gdb ./simple).
  - Platzieren Sie einen Breakpoint auf der Funktion `mistakes1`, 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 `mistakes2`, setzen Sie den Programmlauf fort, welchen Typ hat `buf`?
  - Setzen Sie den Programmlauf fort, welcher Text wird nun 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 in `gdb` die Funktion `mistakes4` 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 dabei wie folgt aussehen:

```
1: 1
2: 2
3: 3
4: 4
5: 5
```

- 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 Programmlaufzeit der Speicher korrekt freigegeben wird. (**Hinweis**: Den Speicher einfach mit `static` zu allokieren ist **nicht** erlaubt. Verzichten Sie darüber hinaus auf globale Arrays und Variablen.)

Dokumentieren Sie alle Fehler bzw Probleme, die zu Abstürzen, Speicherfehlern oder falschem/undefiniertem Verhalten führen. Nutzen Sie hierfür inline Kommentare im Quellcode. Notieren Sie hierfür für jeden vorhandenen Fehler die Code-Zeile(n), welche fehlerhaft sind und den genauen Grund der Ursache (z. B. Speicher mehrfach freigegeben). Sie können die Dokumentation im PDF ablegen oder direkt im Quellcode als Kommentare einfügen.

1.2 Debugging einer komplexeren Anwendung (150 Punkte)

Im Verzeichnis `broken-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 und Ausgabe 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. Achten sie darauf, ob die Terminal-Ausgabe Sinn ergibt. Korrigieren Sie alle Fehler im Programm. Modifizieren Sie hierbei den Code so wenig wie nötig. Bedenken Sie, dass das Programm aus mehreren Dateien besteht.

Hinweis: Das Programm sollte zum Testen wie folgt aufgerufen werden:
`$ ./partdiff-seq 1 1 100 1 2 10`

Abgabe

Abzugeben sind:

1. Ein PDF mit den Fehlerbeschreibungen, falls nicht bereits im Quellcode dokumentiert, sowie die gdb-Ausgaben der ersten Aufgabe

2. Die modifizierten Programme

Achten Sie darauf, dass Ihre Programme ohne Warnungen oder Fehler bauen und Valgrind keine Probleme anzeigt. Packen Sie ein komprimiertes Archiv (`.tar.gz`) aus dem sauberen Verzeichnis (**ohne Binärdateien oder versteckte Dateien**). Benennen Sie das Archiv nach den Nachnamen der Gruppenmitglieder (z. B. `MustermannMusterfrau.tar.gz`).
Ein Mitglied Ihrer Gruppe legt das Archiv dann in

`/home/USERNAME/HR-Abgaben-2627/Blatt-02`

ab und schickt nach erfolgter Abgabe eine E-Mail an `jannek.squar@uni-hamburg.de`, in der Sie einfach den absoluten Pfad zu Ihrem Archiv angeben:

```
/home/USERNAME/HR-Abgaben-2627/Blatt-02/MustermannMusterfrau.tar.gz.
```

Setzen Sie Ihre Gruppenmitglieder auf CC.

Hinweis: Mit `make clean` können Binärdaten, die beim Bauen des Programms erzeugt wurden, entfernt werden.