Portfolioprüfung – Werkstück A – Alternative 1

1) Aufgabe

Entwickeln Sie einen Simulator zur **Speicherverwaltung bzw. Speicherpartitionierung**.

Entwickeln und implementieren Sie einen Simulator für die in der Vorlesung vermittelten Speicherverwaltungssysteme. (statische Partitionierung, dynamische Partitionierung, Buddy). Dieser soll einen Speicher mit einer bestimmten Größe und Konfiguration darstellen und die Zuweisung von Prozessen bzw. Freigabe von Prozessen korrekt simulieren und demonstrieren. Die Größe des Speichers soll der Benutzer (mit sinnvollen Einschränkungen!) frei festlegen können.

Entwickeln und implementieren Sie Ihre Lösung als Bash-Skript oder als C-Programm als freie Software (Open Source) und verwenden Sie hierfür ein Code-Repository, z.B. bei GitHub oder GitLab (siehe Abschnitt 5).

Bearbeiten Sie die Aufgabe in Teams zu maximal **4 Personen** oder **5 Personen** (siehe hierzu Abschnitt 3).

Schreiben Sie eine aussagekräftige und ansehnliche **Dokumentation** (Umfang: **8-10 Seiten**) über Ihre Lösung. Eine Vorlage für das Textsatzsystem LaTeX und weitere Informationen zum Verfassen einer guten Dokumentation finden Sie auf der Vorlesungsseite. Sie müssen die Vorlage nicht zwingend verwenden und Sie müssen auch nicht zwingend LaTeX verwenden. Dieses Projekt ist aber eine sehr gute Gelegenheit sich mit LaTeX auseinander zu setzen und die Vorlage enthält viele hilfreiche Hinweise. In der Dokumentation beschreiben Sie Ihre Lösung (Architektur, etc.), die Aufteilung der Komponenten auf die im Team beteiligten Personen, ausgewählte Probleme und deren Lösungen, etc. Auch einzelne Screenshots können sinnvoll sein. Die Dokumentation soll ein technischer Bericht sein, in der dritten Person formuliert sein, und ganz auf relevante Aspekte Ihrer Lösung fokussieren.

Bereiten Sie einen **Vortrag mit Präsentationsfolien und Live-Demonstration** (Umfang: **15-20 Minuten**) vor. Demonstrieren Sie die Funktionalität der Lösung in der Übung. Eine Vorlage für Präsentationsfolien für das Textsatzsystem LaTeX mit der Erweiterungsklasse `beamer` und weitere Informationen zum Verfassen einer guten Projektpräsentation finden Sie auf der Vorlesungsseite. Auch diese Vorlage müssen Sie nicht zwingend verwenden und Sie müssen auch für die Präsentationsfolien nicht zwingend LaTeX verwenden. Dieses Projekt ist aber eine sehr gute Gelegenheit sich mit LaTeX und der Erweiterungsklasse `beamer` auseinander zu setzen und die Vorlage enthält viele hilfreiche Hinweise. Der Vortrag fokussiert ganz auf Ihre Lösung und nicht auf die Aufgabe oder Inhalte der Vorlesung. Es geht hierbei ausschließlich um Ihre Leistung im Projekt. Es versteht sich von selbst, dass alle im Team beteiligten Personen, ihre selbst entwickelten und implementierten Komponenten

vorstellen und demonstrieren können und natürlich auch Fragen zur Lösung und zu ihren Komponenten beantworten können.

2) Anforderungen an den Simulator

- Das fertige Programm soll eine Kommandozeilenanwendung sein.
- Der Quellcode soll durch Kommentare verständlich sein.
- Benutzer sollen die Werte für den benötigten Speicher der entsprechenden Prozesse interaktiv eingeben können.
- Benutzer sollen die Prozesse und deren benötigten Speicher auch über eine einzulesende Eingabedatei definieren können. Pfad und Dateiname der Eingabedatei definieren die Benutzer per Kommandozeilenargument, also z.B. `-processlistfile <dateiname>`.
- Benutzer sollen Speicher einzelner Prozesse interaktiv wieder Freigeben können.
- Benutzer definieren beim Start des Simulators via Kommandozeilenargument, welches Speicherverwaltungssystem verwendet (simuliert) werden soll, z.B.: `-speicherverwaltung <static/dynamic/buddy>`
- Für die Dynamische Partitionierung soll per Kommandozeilenargument übergeben werden ob First Fit, Next Fit oder Best Fit verwendet wird, z.B.: `-konzept <first/next/best>`
- Realisieren Sie einen Effizienzindex, welcher eine Auswertung von interner und externer Fragmentierung verwendet um in der Ausgabe anzuzeigen wie Effizient die Speicherverwaltung während der Laufzeit arbeiten.
- Zudem soll der Simulator alle Schritte der Speicherverwaltungen in einer Logdatei dokumentieren, deren Pfad und Dateiname die Benutzer per Kommandozeilenargument frei definieren, also z.B. `-logdatei <dateiname>`.
- Fehlerhafte Kommandozeilenargumente soll das Programm erkennen und durch Fehlermeldungen und/oder Abbruch des Programms sinnvoll behandeln können.

3) Erweiterung der Aufgabe für 5 Personen

Damit die Aufgabe sinnvoll von fünf Personen bearbeitet werden kann, verwenden Sie eine geeignete Bibliothek oder eine vergleichbare Lösung, die es ermöglicht, grafisch ansehnliche Ausgaben auf der Kommandozeile zu realisieren. Verwenden Sie dafür eine geeignete Bibliothek oder eine vergleichbare Lösung, die es ermöglicht,

grafisch ansehnliche Ausgaben auf der Kommandozeile zu realisieren. Beispiele für Bibliotheken, die „grafische Darstellung“ und sogar Bedienung in der Shell ermöglichen, sind **ncurses**[1] [2] (für C-Programme), **Termbox**[3] (für C-Programme - wird nicht mehr weiterentwickelt), **dialog**[4] [5] [6] (für Shell-Scripte) oder **Whiptail**[7] [8] [9] (für Shell-Scripte).

4) Beispielhafte Befehle

Beispiel für einen Aufruf des Simulators mit einer Eingabedatei, in der Prozesse definiert sind, deren Abarbeitung gemäß statischer Speicherveraltung simuliert werden soll:

```
$ simulator.sh -speicherverwaltung static \
               -processlistfile inputfile.txt \
               -logdatei logfile.txt
```

Beispiel für einen Aufruf des Simulators mit einer Eingabedatei, in der Prozesse definiert sind, deren Abarbeitung gemäß dynamischer Speicherveraltung mit Best Fit simuliert werden soll:

```
$ simulator.sh -speicherverwaltung dynamic \
               -konzep best \
               -processlistfile inputfile.txt \
               -logdatei logfile.txt
```

Beispiel für einen Aufruf des Simulators für Buddy-Speicherverwaltung ohne eine definierte Eingabedatei – also interaktive Eingabe der Prozesse:

```
$ simulator.sh -speicherverwaltung buddy \
               -logdatei logfile.txt
```

[1] http://openbook.rheinwerk-verlag.de/linux_unix_programmierung/Kap13-002.htm
[2] https://de.wikibooks.org/wiki/Ncurses:_Grundlegendes
[3] https://github.com/nsf/termbox
[4] http://openbook.rheinwerk-verlag.de/shell_programmierung/shell_007_007.htm
[5] https://www.linux-community.de/ausgaben/linuxuser/2014/03/mehr-komfort/
[6] https://linuxkurs.spline.de/Ressources/Folien/Linux-Kurs-7.pdf
[7] https://en.wikibooks.org/wiki/Bash_Shell_Scripting/Whiptail
[8] https://saveriomiroddi.github.io/Shell-scripting-adventures-part-3/
[9] https://www.dev-insider.de/dialogboxen-mit-whiptail-erstellen-a-860990/

5) Einige abschließende Worte zum Code-Repository

Das Code-Repository müssen alle am Team beteiligten Personen erkennbar verwenden, das heißt sie müssen ihre Komponente(n) aktiv entwickeln. Die aktive Mitarbeit bei der Entwicklung und Implementierung des Programms muss für alle Teammitglieder in der Commit-Historie des Code-Repositories erkennbar sein. Kommen einzelne am Team beteiligten Personen nicht in der Commit-Historie des Code-Repositories vor, ist deren Beitrag bei der Implementierung des Programms mehr als unglaubwürdig.

Das Code-Repository soll während der gesamten Projektlaufzeit einsehbar und die kontinuierliche Entwicklung des Simulators erkennbar sein. Stellen Sie hierfür Ihr Repository auf Public und nicht auf Private. Ein oder nur sehr wenige Commits einer einzelnen Person am Ende der Projektlaufzeit sprechen gegen eine gemeinsame Entwicklung und sinnvolle Teamarbeit.

6) Literatur

- Foliensatz 2 der Vorlesung **Betriebssysteme und Rechnernetze** im SS2024
- **Betriebssysteme kompakt**, *Christian Baun*, 3. Auflage, Springer Vieweg, Kapitel 5 „Speicherverwaltung“
- **Operating Systems – Internals and Design Principles**, *William Stallings*, 4. Auflage, Prentice Hall (2001), S. 305-315
- **Betriebssysteme**, *William Stallings*, 4. Auflage, Pearson Studium (2003), S. 365-368
- **Moderne Betriebssysteme**, *Andrew Tanenbaum*, 3. Auflage Pearson Studium (2009), S. 232-240 und S. 883-884