Portfolioprüfung – Werkstück A – Alternative 5

1) Aufgabe

Entwickeln Sie einen Simulator zur **Deadlock-Erkennung mit Matrizen**.

Der Simulator soll für einen Ressourcenvektor überprüfen, ob ein Deadlock vorliegt oder nicht.

Entwickeln und implementieren Sie Ihre Lösung als Bash-Skript, als Python-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.
- Der Bereitstellung des Ressourcenvektors soll interaktiv durch Eingabe möglich sein.
- Zudem soll es möglich sein, einen Ressourcenvektors aus einer lokalen Datei beim Start des Programms einzulesen. Pfad und Dateiname der Eingabedatei definieren die Benutzer per Kommandozeilenargument, also z.B. `-ressourcenvektor <dateiname>`.
- Die Ausgabe des Simulators enthält zur Kontrolle den eingelesenen oder interaktiv eingegebenen Ressourcenvektors sowie die vom Simulator berechnete Belegungsmatrix und die Anforderungsmatrix und den initialen Ressourcenrestvektor.
- Nach jeden Schritt (Auswahl des nächsten Prozesses, etc.) gibt der Simulator den neuen Ressourcenrestvektor aus.
- Kommt der Simulator an einen Punkt, bei dem mehrere Prozesse ausgeführt werden können, soll der Simulator den Benutzer fragen, welcher Prozess als nächstes ausgeführt wird. Dieses interaktive Verhalten kann der Benutzer durch ein entsprechendes Kommandozeilenargument, also z.B. `-noninteractive`, unterbinden. in diesem Fall soll der Simulator bei mehreren möglichen Prozessen zufällig entscheiden, welchen Prozess er als nächstes ausführt.
- Können alle Prozesse ausgeführt werden, ohne dass es zum Deadlock kommt, gibt der Simulator dieses entsprechend aus.
- Kommt es zum Deadlock, gibt der Simulator dieses entsprechend aus.
- Der Simulator soll alle Schritte 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. Die Bibliothek soll die Ausgabe und interaktive Arbeit mit dem Simulator „grafisch“ aufwerten.

Beispiele für Bibliotheken, die „grafische Darstellung“ und Bedienung in der Shell ermöglichen, sind **ncurses**[1] [2] (für C-Programme), **Termbox**[3] (für C-Programme oder Python-Scripte - wird nicht mehr weiterentwickelt), **Textual**[4] (für Python-Scripte), **Typer**[5] (für Python-Scripte), **Asciimatics**[6] (für Python-Scripte), **pyTermTk**[7] (für Python-Scripte), **dialog**[8] [9] [10] (für Shell-Scripte) oder **Whiptail**[11] [12] [13] (für Shell-Scripte).

4) Beispielhafte Befehle

Beispiel für einen Aufruf des Simulators mit einer Eingabedatei, in der der Ressourcenvektor definiert ist. Immer, wenn mehrere Prozesse ausgeführt werden können, soll der Simulator zufällig (nicht-interaktiv) entscheiden, welchen Prozess er als nächstes ausführt:

```
$ simulator.sh -ressourcenvektor inputfile.txt \
               -noninteractive \
               -logdatei logfile.txt
```

Beispiel für einen Aufruf des Simulators mit einer Eingabedatei, in der der Ressourcenvektor definiert ist. Immer, wenn mehrere Prozesse ausgeführt werden können, soll der Simulator den Benutzer interaktiv fragen, welcher Prozess als nächstes ausführt wird:

```
$ simulator.sh -ressourcenvektor inputfile.txt \
               -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] https://www.textualize.io
[5] https://typer.tiangolo.com
[6] https://github.com/peterbrittain/asciimatics
[7] https://ceccopierangiolieugenio.github.io/pyTermTk/
[8] http://openbook.rheinwerk-verlag.de/shell_programmierung/shell_007_007.htm
[9] https://www.linux-community.de/ausgaben/linuxuser/2014/03/mehr-komfort/
[10] https://linuxkurs.spline.de/Ressources/Folien/Linux-Kurs-7.pdf
[11] https://en.wikibooks.org/wiki/Bash_Shell_Scripting/Whiptail
[12] https://saveriomiroddi.github.io/Shell-scripting-adventures-part-3/
[13] https://www.dev-insider.de/dialogboxen-mit-whiptail-erstellen-a-860990/

Beispiel für einen Aufruf des Simulators ohne eine definierte Eingabedatei – also interaktive Eingabe des Ressourcenvektors. Immer, wenn mehrere Prozesse ausgeführt werden können, soll der Simulator den Benutzer interaktiv fragen, welcher Prozess als nächstes ausführt wird:

```
$ simulator.sh -logdatei logfile.txt
```

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 6 der Vorlesung **Betriebssysteme und Rechnernetze** im SS2024
- **Betriebssysteme kompakt**, *Christian Baun*, 3. Auflage, Springer Vieweg, Abschnitt 9.2.3 5 „Verhungern und Deadlock“
- **Betriebssysteme**, *William Stallings*, 4. Auflage, Pearson Studium (2003), S. 325-329
- **Betriebssysteme**, *Rüdiger Brause*, 4. Auflage, Springer (2017), S. 96
- **Betriebssysteme**, *Andrew Tanenbaum*, 3. Auflage, Pearson Studium (2009), S. 523-526