Portfolioprüfung – Werkstück A – Alternative 2

1) Aufgabe

Entwickeln Sie einen Simulator für das **Multilevel-Feedback-Scheduling**.

Der Simulator soll die Ausführungsreihenfolge für eine bestimmte Anzahl an Prozessen berechnen und entweder als Gantt-Diagramm (Zeitleiste) oder in einer anderen geeigneten Form ausgeben. Die Anzahl der Prozesse und deren jeweilige Laufzeiten und Ankunftszeiten kann der Benutzer (mit sinnvollen Einschränkungen!) frei festlegen.

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 **5 Personen**.

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 Anzahl der Warteschlangen und deren Zeitquantum über eine interaktive Benutzereingabe und per Kommandozeilenargumente definieren können. Die Kommandozeilenargumente könnten z.B. folgende Form haben `-warteschlangen 4 -quantum 1 2 5 10`.
- Benutzer sollen die Prozesse über eine einzulesende Eingabedatei definieren. Pfad und Dateiname der Eingabedatei definieren die Benutzer per Kommandozeilenargument, also z.B. `-processlistfile <dateiname>`.
- Jeder Prozess in der Eingabedatei hat drei Attribute:
  - Name (oder alternativ Nummer)
  - CPU-Laufzeit
  - Ankunftszeit
- Die Ausgabe enthält die Ausführungsreihenfolge der Prozesse entweder als Gantt-Diagramm (Zeitleiste) oder in einer anderen geeigneten Form. Verwenden Sie dafür eine geeignete Bibliothek oder eine vergleichbare Lösung, die es ermöglicht, grafisch ansehnliche Ausgaben auf der Kommandozeile zu realisieren und/oder eine Bilddatei generieren.
- Teil der Ausgabe sollen auch die Laufzeiten[1] und Wartezeiten[2] der einzelnen Prozesse sowie die durchschnittliche Laufzeit und durchschnittliche Wartezeit sein.
- Die Ausgabe des Simulators soll in eine Ausgabedatei (eine einfache Textdatei genügt) ausgegeben werden, deren Pfad und Dateiname die Benutzer per Kommandozeilenargument frei definieren, also z.B. `-ausgabedatei <dateiname>`.
- Zudem soll der Simulator alle Schritte beim Scheduling 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.

[1] Zeit [s] von der Prozesserzeugung bis zur Terminierung.
[2] Zeit [s], in der der Prozess im Zustand `bereit` ist, aber keinen Zugriff auf die CPU hat.

3) Anforderungen an die Simulation des Multilevel-Feedback-Scheduling

Innerhalb der Warteschlangen wird **Round Robin** eingesetzt.

Der Simulator implementiert keine Prozessprioritäten oder Zeitmultiplexe für die einzelnen Warteschlangen.

Die Arbeitsweise ist wie folgt: Neue Prozesse werden nach ihrer Erzeugung in Warteschlange 1 eingereiht.

Wird ein Prozess von der CPU verdrängt, wird er in die nächst tiefere Warteschlange eingereiht. Das Vorgehen setzt sich so lange fort, bis ein Prozess in der tiefsten Warteschlange eingereiht ist.

Zuerst werden Prozesse in Warteschlange 1 gemäß ihrer Reihenfolge bearbeitet. Nur wenn Warteschlange 1 keine Prozesse enthält, werden die Prozesse in Warteschlange 2 gemäß ihrer Reihenfolge bearbeitet. Nur wenn die Warteschlagen 1 und 2 keine Prozesse enthalten, werden die Prozesse in Warteschlange 3 gemäß ihrer Reihenfolge bearbeitet, usw.

4) Beispielhafte Befehle

Beispiel für einen Aufruf des Simulators mit Textausgabe:

```
$ simulator.sh -warteschlangen 4 \
               -quantum 1 2 5 10 \
               -processlistfile inputfile.txt \
               -logdatei logfile.txt \
               -ausgabeformat text \
               -ausgabedatei outputfile.txt
```

Beispiel für einen Aufruf des Simulators mit grafischer Ausgabe als Bilddatei:

```
$ simulator.sh -warteschlangen 4 \
               -quantum 1 2 5 10 \
               -processlistfile inputfile.txt \
               -logdatei logfile.txt \
               -ausgabeformat grafisch \
               -ausgabedatei image.png
```

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 5 der Vorlesung **Betriebssysteme und Rechnernetze** im SS2024
- **Betriebssysteme kompakt**, *Christian Baun*, 3. Auflage, Springer Vieweg, Kapitel 5 „Speicherverwaltung“
- **Betriebssysteme**, *William Stallings*, 4. Auflage, Pearson Studium (2003), S. 466-477
- **Betriebssysteme**, *Rüdiger Brause*, 4. Auflage, Springer (2017), S. 39-64
- **Betriebssysteme**, *Eduard Glatz*, 4. Auflage, dpunkt.verlag (2019), S. 160-170
- **Betriebssysteme**, *Carsten Vogt*, 1. Auflage, Spektrum Akademischer Verlag (2001), S. 65-67