Übungsblatt 7: Weitere Aufgaben

1 Skizziere kurz einige Probleme des nebenläufigen Zugriffs auf Betriebsmittel.

- Inkorrektes Verändern von gemeinsamen Daten (z. B. nebenläufiges Lesen → unabhängiges lokales Verändern → nebenläufiges Schreiben)
- Durcheinanderschreiben auf Druckern
- Verklemmungen bei wechselseitigem Warten auf exklusive Betriebsmittel

2 Grenze die Begriffe *Nebenläufigkeit*, *Quasi-Parallelität* und *Parallelität* voneinander ab. Was verstehen wir unter *Nichtdeterminismus*?

Nebenläufigkeit: Nicht-sequentielle Ausführung mehrerer Prozesse/Threads auf einer oder mehreren CPUs, so dass der Beginn der einen Ausführung vor dem Ende der anderen liegen kann (d.,h. Verschachtelung oder Parallelität der Ausführungen möglich).

Parallelität: Gleichzeitige Ausführung mehrerer Prozesse/Threads durch mehrere CPUs.

- Quasi-Parallelität: Nebenläufigkeit, aber keine Parallelität.
- Nichtdeterminismus: Ergebnis und/oder Abfolge des Programms nicht vorherbestimmbar (auch bei gleicher Eingabe).

3 Welche Nebenläufigkeitseigenschaften bzw. -probleme werden durch die drei folgenden „klassischen“ Szenarien ausgedrückt:

***Erzeuger/Verbraucher (Producer/Consumer)*:** Verbraucher muß warten, bis Erzeuger die erforderlichen Voraussetzungen geschaffen und mitgeteilt hat. Klassischer Fall von einseitiger Synchronisation.

***Leser/Schreiber (Reader/Writer)*:** Synchronisationsprobleme tauchen nur beim Verändern von Informationen auf. Also: *Ein* Schreiber alternativ zu beliebig vielen *nebenläufigen* Lesern.

***Speisende Philosophen (Dining Philosophers)*:** Fünf Philosophen wechseln jeweils zwischen den beiden möglichen Zuständen *Denken* und *Essen*/ Zum Essen setzen sie sich an einen runden Tisch, auf dem fünf Stäbchen angeordnet sind, von denen sie zwei benachbarte benötigen. Als Folge können nur zwei Philosophen gleichzeitig essen. Implementierungen, die zulassen, dass sich alle fünf Philosophen quasi gleichzeitig an den Tisch setzen und ihr erstes Stäbchen aufnehmen, können in eine Verklemmung laufen (geschachtelte kritische Abschnitte).

4 Was ist ein Thread? Skizziere ein sinnvolles Anwendungsbeispiel für die Verwendung mehrerer Threads innerhalb eines Prozesses.

Thread: Kontrollfaden innerhalb eines Prozesses, der ein angegebenes Programmstück unabhängig von den anderen Threads dieses Prozesses durchläuft.

Beispiele:

- Nebenläufige Ausführung mehrerer Alternativen einer komplexen Berechnung.
- Mehrere identische „Agenten“ bieten jeweils eine Benutzerschnittstelle zu einer gemeinsamen Datenbank.
- Mehrere Komponenten eines komplexen Editors (Eingabebearbeitung, Formatierung, Garbage Collection, ...)

5 Grenze den Thread-Begriff gegen den UNIX-Prozess-Begriff ab (Adressraum, Zustandsinformationen usw.). Was haben *Light-Weight-Prozesse (LWPs)* damit zu tun?

Thread vs. Prozess: Jeder Thread hat einen eigenen Ausführungszustand (z. B. Registerinhalte), aber einen gemeinsamen Adreßraum mit allen anderen Threads des Prozesses.

Light Weight Process (LWP): sind Threads aus der Sicht des UNIX-Kerns. Scheduling findet auf der Grundlage von LWPs statt. Damit (in einem Einprozessorsystem) wirkliche Nebenläufigkeit innerhalb eines Prozesses erzielt werden kann, müssen die User-Level-Threads also auf LWPs abgebildet werden.