Übungsblatt 8: Weitere Aufgaben

1 Die Routinen `pthread_create()`, `pthread_join()`, `pthread_exit()` realisieren die Erzeugung und Termination von Threads in der UNIX-Multithreading-Umgebung. Vergleiche ihre Funktionalität mit den Systemaufrufen zur Erzeugung und Termination von Prozessen (`wait()`, `fork()` und `exit()`). Warum arbeitet `pthread_create()` deutlich anders als `fork()`?

`thr_create()` ist ähnlich zu `fork()`. Jedoch wird keine Kopie des Adressraums angelegt, da es sich immer noch um denselben Prozess handelt.

`thr_join()` entspricht `wait()`. `thr_exit()` entspricht `exit()`. Jedoch haben die Threads kein „Leben“ über die Lebenszeit des Prozesses hinaus.

2 Was versteht man unter einseitiger bzw. mehrseitiger Synchronisation? Gib jeweils ein Anwendungsbeispiel an.

Einseitige Synchronisation: Prozess/Thread wartet auf „Signal“ (Meldung) eines anderen und kann erst danach weiterlaufen. Beispiel: Erzeuger-/Verbraucher-Problem.

Mehrseitige Synchronisation: Mehrere Prozesse/Threads greifen auf gemeinsamen kritischen Abschnitt zu. Zur Realisierung des gegenseitigen Ausschlusses darf (zu einem Zeitpunkt) nur ein Prozess im kritischen Abschnitt sein, die anderen müssen solange davor warten. Beispiel: Mehrere Schreiber auf einer gemeinsamen Datenstruktur.

3 Was ist ein kritischer Abschnitt? Wie kann man den gegenseitigen Ausschluss gewährleisten? Warum ist ein Unterbrechungsausschluss dabei nicht immer das geeignete Mittel?

Kritischer Abschnitt: Programmstück mit Zugriff auf gemeinsame Betriebsmittel/Daten.

Gegenseitiger Ausschluss: Zu einem Zeitpunkt darf nur ein Prozess/Thread in den kritischen Abschnitt hinein, wird in der Regel durch *Locking-Protokoll* realisiert.

Probleme mit Unterbrechungsausschluss:

- Unterbrechungen müssen oft dringend behandelt werden und dürfen daher nicht zu lange blockiert werden.

- Normaler Unterbrechungsausschluss funktioniert bei Multiprozessoranlagen nicht, da er nur für die lokale CPU gilt.

4 Nach welchen Kriterien wird die Korrektheit bzw. Güte von Locking-Algorithmen bewertet? Wie geht man dabei vor?

Kriterien für Locking-Algorithmen:

- Sichern sie den gegenseitigen Ausschluss?
- Sind sie verklemmungsfrei?
- Vermeiden sie *Starvation* bzw. *After-you-after-you*?

Vorgehen: Runterbrechen des Algorithmus auf unteilbare Operationen. Im Prinzip durchspielen und bewerten aller möglichen Ausführungsreihenfolgen (bzw. plausibilisieren der möglichen Fälle).

5 Warum sollte man die Bewertung von Locking-Algorithmen auf der Grundlage von unteilbaren Operationen durchführen?}

Weil nur dann alle potentiellen Nebenläufigkeiten berücksichtigt worden sein können.

6 Auf welche verschiedene Arten kann man Verklemmungen angehen? Wie arbeitet der Bankiersalgorithmus?

Behandlung von Verklemmungen durch:

- *Ignorieren*: Verklemmung kommt selten vor, dann zur Not neu booten.
- *Erkennen und Beheben*: Liegt ein Zyklus vor? Dann solange Prozesse daraus entfernen, bis es wieder geht.
- *Verhindern*: Verklemmungen nicht zulassen, indem man die dafür erforderliche Kombination von Randbedingungen verhindert. In der Regel: Zyklen nicht zulassen, z. B. durch geordnete Zuteilung von Betriebsmitteln nach irgendeinem Nummernschema.
- *Vermeiden*: keine unsicheren Zustände betreten, z. B. durch Verwendung des Bankiersalgorithmus.

Bankiersalgorithmus: Von sicherem Zustand ausgehend nur sichere Folgezustände betreten: Zuteilung nur, wenn auch danach noch genug Betriebsmittel vorhanden, um alle Prozesse (ggf. in bestimmter Reihenfolge) zu Ende führen zu können. Dazu Wissen über maximale Anforderung von Betriebsmitteln durch Prozesse erforderlich.