Koordination & Ausschluss II

Tutorium 06

Betriebssysteme SS2024

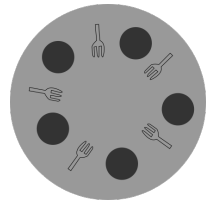
Inhaltsverzeichnis

1 Das Philosophenproblem

2 Spinlocks

3 Semaphore

Das Philosophenproblem



- Fünf Philosophen sitzen an einem runden Tisch mit fünf Tellern und fünf Gabeln
- Jeder Philosoph kann entweder denken oder essen.
- Um zu Essen nimmt jeder Philosoph zuerst die linke und dann die rechte Gabel neben seinem Teller
- Wenn ein Philosoph fertig ist mit Essen legt er beide Gabeln zurück

Das Philosophenproblem (2)

- Sollten sich drei oder mehr Philosophen **gleichzeitig** dazu entscheiden zu essen, so kann das zu einem **Deadlock** führen
- Ein Deadlock ist ein Zustand wenn eine zyklische Wartesituation auftritt
- Jeder beteiligte Prozess wartet auf die Freigabe von Ressourcen die ein anderer Prozess exklusiv belegt

Ansätze um Deadlocks zu verhindern

Erster möglicher Ansatz:

Ansätze um Deadlocks zu verhindern

Erster möglicher Ansatz:

- Nicht zulassen, dass alle Philosophen gleichzeitig sitzen und essen/denken
- Bei höchstens vier Philosophen am Tisch ist garantiert, dass ein Philosoph beide Gabeln aufnehmen kann

Zweiter möglicher Ansatz:

Ansätze um Deadlocks zu verhindern

Erster möglicher Ansatz:

- Nicht zulassen, dass alle Philosophen gleichzeitig sitzen und essen/denken
- Bei höchstens vier Philosophen am Tisch ist garantiert, dass ein Philosoph beide Gabeln aufnehmen kann

Zweiter möglicher Ansatz:

- Das Aufnehmen einer beliebigen Gabel markiert den Eintritt in einen kritischen Abschnitt
- Solange sich ein Philosoph im kritischen Abschnitt befindet dürfen andere Philosophen keine Gabeln mehr aufnehmen
- Der kritische Abschnitt wird wieder verlassen sobald ein Philosoph beide Gabeln aufgenommen hat

Spinlock

Definition: Ein Spinlock ist eine Sperre zum Schutz gemeinsam genutzter Ressourcen, welche mittels aktivem Warten umgesetzt wird.

Vorteil:

- Für kurze kritische Abschnitte effizient

Nachteile:

- Aktives Warten verbraucht Rechenzeit
- Effizienz abhängig von der Scheduling-Strategie

Spinlock (2)

Funktionsweise

- Zu Beginn wird die Sperrvariable auf den Wert *frei* gesetzt
- Wenn Sperrvariable *frei* ist, kann sie von einem Prozess *gesperrt* werden und der Prozess kann auf die gesicherte Ressource zugreifen
- Nach der Modifizierung der Ressource wird die Sperrvariable wieder auf *frei* gesetzt
- Ist die Sperrvariable auf *gesperrt* gesetzt, so überprüft ein wartender Prozess diese ständig bis sie *frei* ist

Semaphore

Definition: Ein Semaphor ist eine Datenstruktur die eine Variable verwaltet, welche angibt wie viele Prozesse gleichzeitig auf eine Ressource zugreifen können. Die Variable wird von den folgenden zwei Operationen modifiziert:

- P-Operation (Probieren): dekrementiert den Wert der Variablen
- V-Operation (Freigeben): inkrementiert den Wert der Variablen

Semaphore (2)

Funktionsweise:

- Vor dem Zugriff auf eine geschützte Ressource, muss ein Prozess die P-Operation aufrufen
- Ist der Wert der Variablen ≤ 0, so wird der aufrufende Prozess blockiert und in eine Warteschlange eingereiht
- Falls nicht, bekommt der Prozess Zugriff auf die Ressource
- Zum Freigeben der Ressource muss die V-Operation aufgerufen werden, welche die blockierten Prozesse aus der Warteschlange benachrichtigt

Semaphore (3)

Vorteile:

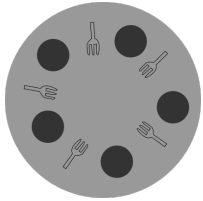
- Kein Aktives Warten
- Scheduler kann in die Semaphor-Operation mit eingebunden werden (z.B. Priorisierung welcher Prozess aufgeweckt wird bei V-Operation)

Nachteil:

- Ineffizient bei kurzen kritischen Abschnitten

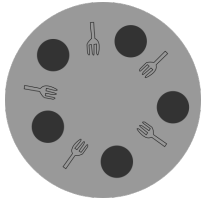
Semaphore werden häufig zur Koordinierung asynchroner Abläufe genutzt

Semaphore (4)



Wie lässt sich das Philosophenproblem mit Semaphoren lösen?

Semaphore (4)



Mögliche Lösung:

- Initialisierung des Semaphor mit 1
- P-Operation ausführen um die zwei Gabeln um den Teller aufzunehmen
- V-Operation zur Freigabe des kritischen Bereichs