21. Synchronisation im Linux-Kernel (5 Punkte)

Die Folien 62–81 (Foliensatz 5) geben eine Übersicht über Synchronisationsmittel, die der Kernel für den Schutz seiner eigenen Datenstrukturen verwendet. Wir besprechen das Thema nicht in der Vorlesung, sondern nur im Rahmen dieser Übung.

a) Der Kernel bietet Funktionen, die **atomar** eine Variable erhöhen (`x++`), erniedrigen (`x--`) oder setzen (`x = Wert`) können. Folie 65 zeigt diese Funktionen (`atomic_inc`, `atomic_dec`, `atomic_set`). „Atomar“ bedeutet dabei, dass eine solche Operation nicht unterbrochen werden kann, also während ihrer Ausführung kein Context Switch möglich ist. Auch beim Einsatz mehrerer Prozessoren können sich nicht zwei auf verschiedenen CPUs laufende Kernel-Tasks „in die Quere“ kommen und echt parallel eine solche atomare Funktion mit derselben Variablen ausführen.

– Wozu wird ein solches Feature im Kernel benötigt?

– Inwiefern hängt auch das Funktionieren von Mutexen, Semaphoren etc., die normale Anwenderprogramme nutzen, davon ab, dass der Kernel diese Funktionen bereitstellt?

– Warum verwendet der Kernel nicht einfach die User-Level-Mutexe, sondern eigene Methoden?

b) Auf Folie 69 wird u. a. die Funktion `test_and_set_bit()` vorgestellt. Wie könnten Sie diese verwenden, um im Kernel einen kritischen Bereich zu schützen?

c) **Spin Locks** (Folie 70-73) funktionieren ähnlich wie Mutexe, allerdings mit aktivem Warten: Daher kommt auch ihr Name (spinning lock).

Warum verwendet man an manchen Stellen im Code Spin Locks anstelle von klassischen Mutexen? (Die Antwort steht in den genannten Folien.) Überlegen Sie sich ein Beispiel für eine Aufgabe, die der Kernel hat und bei der Sie ein Spin Lock erwarten würden.

d) Folien 74/75 beschreiben die sog. **Reader-Writer-Locks**. Erklären Sie, wie diese sich von normalen Spin Locks unterscheiden.

e) Ab Folie 76 werden **Kernel-Semaphore** beschrieben (nicht mit User-Space-Semaphoren zu verwechseln).

– Nennen Sie die wichtigsten Eigenschaften eines Kernel-Semaphors.

– Die Funktionen `wait()` und `signal()`, die Sie von User-Space-Semaphoren kennen, heißen bei Kernel-Semaphoren `down()` und `up()`; `down()` zählt also wie `wait()` 1 runter, `up()` zählt wie `signal()` 1 rauf. Erklären Sie, in welcher Weise das Verhalten der Funktion `down_trylock()` von dem der (normalen) Funktion `down()` abweicht.