7 Übungsaufgabe #7: Betriebsmittelzugriff

In der vorherigen Aufgabe wurde der einfache, preemptive Scheduler zu einem Round-Robin Scheduler mit globaler Readyliste. Damit können nun Betriebsmittel verwaltet werden, die durch verschiedene Aktivitätsträger im Wettstreit miteinander stehen.

7.1 Scheduling

Um Betriebsmittel zu verwalten, muss der Scheduler davon Kenntnis haben, welche Prozesse welche Betriebsmittel belegen, bzw. darauf warten. Dies kann entweder in Form von Tabellen oder Listen geschehen. In dieser Aufgabe soll Ihr Scheduler so erweitert werden, dass Fäden Betriebsmittel anfordern können. Sind diese belegt oder nicht verfügbar, sollen sie nicht eingeplant werden. Die Testprogramme in dieser Aufgabe werden dieses Merkmal benutzen um konsumierbare Betriebsmittel (Signale) zu produzieren bzw. konsumieren.

7.2 Semaphor

Aus den Vorlesungen „Betriebssysteme“ sowie „Softwaresysteme“ bzw. „Systemprogrammierung“ kennen Sie bereits das Konzept eines Semaphors. Ein solcher Semaphor soll nun im Kontext einer Multiprozessormaschiene implementiert werden. Hierbei dürfen Sie zur Implementierung *spinlocks*, wie sie in der letzten Übungsaufgabe vorgestellt wurden, verwenden um die Warteliste gegen Nebenläufigkeit von anderen Prozessoren zu schützen.

7.3 Optional: Sperrfreier Semaphor

In der Vorlesung am 13. Juli wird ihnen eine Implementierung eines sperrfrei implementierten Semaphor vorgeführt. Diese Lösung kommt anders als die klassische Implementierung ohne Spinlocks aus in dem sie nebenläufige Modifikationen (Einfügen, Entfernen, etc.) toleriert. Integrieren sie diese Techniken in ihre Implementierung.

7.4 Testprogramme

Um Ihre Implementierung zu testen werden zwei Arten von Programmen benutzt: Produzenten (Operation „V“) produzieren Signale an Konsumenten (Operation „P“), welche diese Signale konsumieren. In allen Testfällen bekommen sowohl die Produzenten als auch die Konsumenten eine feste Anzahl an Signalen die konsumiert bzw. produziert werden sollen als Parameter übergeben.
Erweitern Sie ihre Fadenimplementierung derart, dass sich Fäden sauber beenden können. Die Applikationsprozessoren sollen sich herunterfahren, wenn keine aktiven Fäden mehr existieren. Das Beenden geschieht durch ein `return` aus der `main()` Funktion für den jeweiligen Prozessor. Der Bootprozessor soll in diesem Fall auf die Terminierung der Applikationsprozessoren warten und sich dann ebenfalls beenden.

Aufgaben:

→ **Testfall 1**: Starten sie $CPU/2$ Produzenten, welche jeweils n Signale produzieren. Gleichzeitig sollen $CPU/2$ Konsumenten gestartet werden, welche ebenfalls jeweils n Signale konsumieren. Der Testfall ist erfolgreich, wenn alle Programme erfolgreich terminieren.

→ **Testfall 2**: Starten sie ebenfalls $CPU/2$ Produzenten und Konsumenten. Zusätzlich soll eine Unterbrechungsbehandlung, die von einem Zeitgeber periodisch aufgerufen wird, weitere Signale produzieren. Zählen Sie die Anzahl der von der Unterbrechungsbehandlung zusätzlich erzeugten Signale mit. Der Testfall ist erfolgreich, wenn der Zähler in der Semaphore identisch mit der Anzahl der zusätzlich erzeugten Signale aus der Unterbrechungsbehandlung ist.

→ **Optional**: Alle Prozessoren geben unmittelbar vor Beendigung ihre verwendete Rechenzeit (user time) aus. Unter Linux $>$ 2.6.26 können Sie diese mit dem Systemaufruf `getrusage(2)` im Modus `RUSAGE_THREAD` ermitteln.

7.5 Abgabe: am 16.07.2009