5 Übungsaufgabe #5: Fäden und Fadenwechsel

In den vorigen Aufgaben wurde eine Unterbrechungsbehandlung implementiert, mit der die aus der Betriebssystemvorlesung bekannten Hardwareunterbrechungen durch POSIX Signale emuliert werden können.

5.1 Kontrollfäden und Kontextwechsel

Um mehr als einen Aktivitätsträger auf einem Prozessor laufen zu lassen, muss das Betriebsmittel CPU virtualisiert werden. Hierzu wurde bereits eine Coroutinenimplementierung in den Übungen zur Lehrveranstaltung Betriebssysteme für Einprozessorsysteme vorgestellt und implementiert. Zur Erinnerung: Dort wurde streng zwischen der Umschaltung (engl. *dispatching*) und Fadeneinplanung (engl. *scheduling*) unterschieden. Für Mehrkernsysteme können die Techniken zur Fadenumschaltung mit wenigen Anpassungen übernommen werden.

5.2 Multiprozessor Scheduling

In dieser Aufgabe wird ein prototypischer, *Round-Robin*-artiger Scheduler implementiert. Zur Vereinfachung werden Prozesse in dieser Aufgabe nicht von einer CPU auf eine andere migriert, sondern werden auf einer zum Erstellungszeitpunkt festgelegten CPU gehalten. Dies wird in den folgenden Aufgaben erweitert werden. Die eigentliche Umschaltung erfolgt preemptiv aus dem Timerinterrupt der letzten Aufgabe heraus.

5.3 Primitiven zur Ausgabe

Damit die Implementierung getestet werden kann, soll ein erster Systemaufruf zur Ausgabe bereitgestellt werden. Dieser Systemaufruf hat die Signatur `void _printxy(int x, int y, char *fmt, ... )`, und besitzt eine ähnliche Semantik wie `printf(3)`. Zusätzlich erhält er noch Koordinaten, so dass der Benutzer festlegen kann, an welcher Stelle des Bildschirms die Ausgabe erfolgen soll. Zur Implementierung können Sie auf ANSI-Escape-Sequenzen zurückgreifen.
Ihre Testanwendung soll in Abhängigkeit der CPU und der Fadenkennung an einer bestimmten Position eine wechselnde Ausgabe machen. Dadurch soll der Fortschritt z.B. einer Berechnung simuliert werden.

5.4 Systemsynchrone Systemaufrufe

Eine naive Implementierung des Systemaufrufs `_printxy(...)` hat zur Folge, dass nebenläufige Aufrufe dieser Funktion zu Ausgaben an falscher Stelle des Bildschirms führen können. Um dieses Problem in den Griff zu bekommen müssen die eigentlichen Ausgaben in der Funktion sequenzialisiert erfolgen. In der Vorlesung wurde Ihnen die Technik der Fortsetzung (engl. *Continuation*) vorgestellt, die Sie im letzten Teil der Aufgabe am Beispiel von `_printxy(...)` implementieren.

Aufgaben:

→ Implementieren Sie eine Koroutine und einen passenden Dispatcher zur Umschaltung. Die Koroutinen sollen dabei zur Laufzeit erstellt und zerstört werden können. Der Kontext soll im Gegensatz zur Betriebssystemvorlesung nicht in einen Prozesskontrollblock, sondern komplett auf den Stack gesichert werden.

Der Interrupt soll nur auf dem Bootprozessor aktiv sein. Seine Frequenz soll statisch per *kconfig* einstellbar sein. Jeder Interrupt soll nur auf einer CPU den aktuellen Task umschalten um zu vermeiden, dass auf zwei CPUs nebenläufig ein Taskwechsel stattfindet. Der folgende Interrupt soll dann den aktuellen Task auf der „nächsten“ CPU umschalten, so dass lediglich ein Timer benötigt wird.

→ Implementieren Sie einen einfachen, preemptiven *Round-Robin*-artigen Scheduler mit prozessorlokaler Readyliste. Eine Migration von Fäden auf eine andere CPU ist in dieser Aufgabe (noch) nicht gefordert.

→ Implementieren sie einen *Systemaufruf* `_printxy(int x, int y, char *fmt, ... )` zur Ausgabe an einer festen Position.

→ Schreiben Sie ein Testprogramm, welches den Fadenwechsel auf allen Prozessoren demonstriert. Auf jeder CPU sollen dabei (mindestens) zwei Testprogramme gestartet werden.

→ Synchronisieren sie die Ausgaben der `_printxy(...)`-Primitive, so dass die Ausgaben nicht durch nebenläufige Abarbeitung zerstört werden.

Hinweise:

- Es bietet sich an, in der `_printxy(...)` Primitive gepufferte Ausgaben zu verwenden. Dadurch verringert sich die Dauer des kritischen Abschnitt erheblich.

5.5 Abgabe: am 25.06.2009