4 Übungsaufgabe #4: 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.
In dieser Aufgabe soll nun darauf basierend Unterstützung für präemptiven Mehrprozessbetrieb implementiert werden.

4.1 Kontrollfäden und Kontextwechsel

Um mehr als einen Aktivitätsträger auf einem Prozessor laufen zu lassen, muss das Betriebsmittel CPU virtualisiert werden. Hierzu sollt ihr für die `amd64`-Architektur (alle Rechner in der Manlobbi sind 64-bittig) einen Kontextwechsel entweder in Form eines eingebetteten Programms oder als Unterprogramm implementieren (*vgl. Vorlesung vom 5. Juni*).

4.2 Präemptives Multiprozessor Scheduling

In dieser Aufgabe sollt ihr einen *Round-Robin*-artigen Scheduler implementieren. Zur Vereinfachung werden Prozesse in dieser Aufgabe nicht von einer CPU auf eine andere migriert, sondern werden auf einer zum Erstellungszeitpunkt festgelegten CPU gehalten. Die eigentliche Umschaltung erfolgt präemptiv aus dem Timerinterrupt heraus.

4.3 Anpassung der CPUID-Bestimmung

Damit auch weiterhin eine effiziente Bestimmung der (virtuellen) CPUID ohne Auslösen eines Systemaufrufes möglich ist, sollt ihr die CPUID-Bestimmung so anpassen, dass diese auch mit den durch die Koroutinenumschaltung wechselnden Stapeln zurecht kommt.

4.4 Schutz der präemptiven Fadenumschaltung

Beim präemptiven Fadenwechsel kann es zu einer *Race Condition* kommen. In der Vorlesung wurde vorgestellt, wie sich dies ohne Verwendung einer Sperrvariable vermeiden lässt. Dazu wird der *Active*-Zeiger auf den Stapelzeiger einer Koroutine abgebildet. Implementiert dies und mach euch klar unter welchen Umständen dies als Synchronisation für den präemptiven Kontextwechsel auf einem Mehrkernsystem genügt?

4.5 Primitiven zur Ausgabe

Damit die Implementierung besser getestet werden kann, soll die Ein/Ausgabebibliothek aus Aufgabe 2 um eine Ausgabefunktion erweitert werden, bei der der Benutzer die Position am Bildschirm angeben kann, an der die Ausgabe erfolgen soll. Zur Implementierung könnt Ihr auf ANSI Escape-Sequenzen zurückgreifen.
Die 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.

Aufgaben:

→ Implementiert eine Koroutine und einen passenden Dispatcher zur Umschaltung. Die Koroutinen sollen dabei zur Laufzeit erstellt und zerstört werden können.

Der Timerinterrupt sollt über alle virtuellen Prozessoren verteilt werden, so dass man nur eine einzige Zeitgeberquelle benötigt.

→ Implementieren Sie einen einfachen, präemptiven *Round-Robin*-artigen Scheduler mit prozessorlokaler Bereitliste.

→ Passt die effiziente Bestimmung der CPUID an die neuen Gegebenheiten an

→ Schützt die präemptive Fadenumschaltung durch Abbildung des *Active*-Zeigers auf den Stapelzeiger

→ Erweitert eure Ausgabebibliothek um eine weitere Funktion, bei der man die Position des Cursors angeben kann.

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

Hinweise:

- Das ABI zur `amd64`-Architektur findet ihr unter `http://www.x86-64.org/documentation/abi.pdf`. Dort ist unter anderem beschrieben wie die Aufrufkonventionen für Funktionen und die Aufteilung der Register in flüchtige und nicht flüchtige Register geregelt sind.
- Die Unterstützung von Fäden mit Gleitkommazahlenberechnungen ist optional.
- Mit Hilfe von ANSI Escape-Sequenzen kann man das Terminal steuern und so z.B. den Cursor umsetzen, so dass Ausgaben an anderen Stellen im Terminalfenster erscheinen. Eine schöne Übersicht gibt es unter `http://ascii-table.com/ansi-escape-sequences.php`

4.6 Abgabe: am 11.07.2012