1 Übungsaufgabe #1: SMP-fähige Gastebene

Ziel dieser Aufgabe ist es, eine *abstrakte Maschine* zu implementieren, die oberhalb eines vorhandenen Betriebssystems wie **Linux**, **Windows**, **MacOSX** oder **FreeBSD** ausgeführt wird. Dabei wird besonderer Wert auf die *minimale Abstraktion* eines allgemeinen Mehrprozessorsystems gelegt.
Das Ergebnis dieser Aufgabe wird die Basis der folgenden Übungsaufgaben sein.

1.1 Minimale Abstraktion eines Mehrprozessor-Systems

Unter *minimaler Abstraktion* wird eine abstrakte Maschine verstanden, die mit möglichst wenig Aufwand auf möglichst viele verschiedene Plattformen und Betriebssysteme portiert werden kann. Hierzu bietet es sich an die API der abstrakten Maschine minimal zu halten.

1.2 Abstrakte Maschinen als Bibliothek

Die abstrakte Maschine soll im Verlauf der Übung zu einem Fädenangebot (engl. *threads package*) erweitert werden. Dies soll in Form einer Bibliothek geschehen, damit vorhandene Anwendungen durch erneutes Binden weiterverwendet werden können.
Machen Sie sich hierbei Gedanken über die verwendeten Bezeichner von exportierten Symbolen wie Funktionen und globalen Variablen, um *Kollisionen* mit Anwendungsprogrammen zu vermeiden.
Die exportierte API der fertigen Bibliothek soll eine Untermenge der POSIX Thread API (`libpthread`) anbieten.

1.3 Virtuelle Prozessoren

Ein typisches SMP-System besteht aus einem ausgezeichnetem Prozessor, im Folgenden *Bootprozessor* genannt und einem oder mehreren *Applikationsprozessor*en. Der *Bootprozessor* übernimmt beim Systemstart (der Initialisierung der Bibliothek) eine Sonderrolle. Er ist dafür zuständig das System zu initialisieren und die *Applikationsprozessor*en zu starten. Danach führt der Bootprozessor die Hauptfunktion der Anwendung (in `C` die Funktion `main()`) aus, während die Applikationsprozessoren auf auszuführende Tasks warten.
Im Rahmen dieser Aufgabe sollen diese virtuellen Prozessoren mit Hilfe der Gastebene auf *reale* Prozessoren abgebildet werden, damit *echte* Parallelität zwischen den virtuellen Prozessoren beobachtet werden kann.

1.4 Die Hardwareabstraktionsschicht (HAL)

Die Bibliothek wird in *Schichten* entworfen. Die unterste Schicht wird dabei die *Hardwareabstraktionsschicht* genannt und implementiert alle betriebssystem- und plattformabhängigen Funktionen. Damit soll erreicht werden, dass nur diese Schicht angepasst werden muss, wenn die Bibliothek (in einer späteren Aufgabe) auf ein anderes Betriebssystem und/oder Plattform portiert werden soll.
Für diese Übung wird dabei die Verwendung einer evtl. vorhandenen `pthread` Bibliothek untersagt. Stattdessen soll die Funktionalität des jeweils verwendeten Betriebssystems direkt verwendet werden. Für Linux bieten sich dabei die Systemaufrufe `clone(2)` zum Erzeugen eines neuen Aktivitätsträgers, `sched_setaffinity(2)` zum Binden eines Aktivitätsträgers an eine *reale CPU* und `getcpu(2)` zur Identifizierung der aktuellen CPU an.

1.5 Testen der Implementierung

Um die Implementierung zu testen, programmieren Sie eine Schleife, die auf jedem Applikationsprozessor sowie auf dem Bootprozessor die jeweilige Prozessornummer ausgibt. Diese Standardimplementierung einer Startroutine wird in den nächsten Aufgaben erweitert.

Aufgaben:

→ Identifikation einer minimalen API der abstrakten Mehrprozessormaschine mit Begründung der Notwendigkeit jeder Funktion.

→ Implementierung dieses HALs für ein existierendes Betriebssystem (z.B. Linux, Windows, FreeBSD, MacOSX, MPStubs, Xen, KVM, lguest, etc.).

→ Entwickeln einer Demonstrationsanwendung (vgl. 1.5)

Hinweise:

- Es soll ein SMP System simuliert werden, das heißt, dass alle Prozessoren den **selben** (virtuellen) Adressraum teilen!
- Es ist zweckmäßig, eine Funktion `int boot_cpus(void (*fn)(void), int maxcpus)` vorzusehen, welche die Aufgabe hat die Applikationsprozessoren zu starten. Diese Funktion bekommt einen Funktionszeiger `*fn` übergeben, welcher die Startroutine der Applikationsprozessoren referenziert. Der Parameter `maxcpus` begrenzt die Anzahl der zu simulierenden Applikationsprozessoren.
- Obwohl sich alle Prozessoren den selben Adressraum teilen, benötigen alle virtuellen CPUs einen eigenen Stapel (engl. *Stack*)! Die Funktion zum Starten der Applikationsprozessoren muss daher Speicher bereitstellen und dafür sorgen, dass die Funktion `*fn` auf den jeweiligen Applikationsprozessoren mit jeweils eigenen Stapeln ablaufen.
- Unter Linux lässt sich Stapelspeicher mit dem Systemaufruf `mmap(2)` am Besten anfordern. Im Gegensatz zur Bibliotheksfunktion `malloc(3)` wird so die Verwendung der Speicherverwaltung der `libc` vermieden.
- Obwohl die Funktion `*fn` auf allen Applikationsprozessoren dieselbe ist, soll diese aus Gründen der Nachvollziehbarkeit für die Testanwendung trotzdem unterschiedliche Ausgaben erzeugen. Zweckmäßig wäre hier zum Beispiel die Ausgabe der Prozessornummer.

1.6 Abgabe: am 06.05.2009

Die Implementierung der Aufgabe erfolgt in der Programmiersprache `C++`, wobei eine objektorientierte Implementierung nicht zwingend erforderlich ist.