1 Übungsaufgabe #2: Prozessorlokale Variablen

Ziel dieser Aufgabe ist es, den in Aufgabe 1 implementierten *virtuellen* Prozessoren *lokale* Variablen zur Verfügung zu stellen. Hierbei sollen für speziell definierte *globale* Variablen jeweils eine Instanz dieser Variable pro CPU erzeugt werden. Beim Zugriff auf eine Variable soll transparent für den Aufrufer die jeweils dem aktuellen Prozessor zugeordnete Instanz dieser Variable genutzt werden.

1.1 Optimierte Bestimmung der Prozessornummer

Bisher wird der Aufruf von `__get_cpu_id()` auf einen Systemaufruf abgebildet. Dies hat zur Folge, dass die Abfrage der aktuellen Prozessornummer eine teure Operation ist, welche man nicht bei jedem Zugriff auf eine Variable wiederholen möchte. Das Ziel dieser Teilaufgabe ist es, die Bestimmung der Prozessornummer ohne einen Systemaufruf durchzuführen, indem diese über den aktuellen Wert des Stapelzeigers (*x86*: `%esp`) ermittelt wird.

1.2 Prozessorlokale Variablen

Die Implementierung von prozessorlokalen Variablen kann in drei Unteraufgaben geteilt werden. Die erste Aufgabe befasst sich mit der *globalen* Definition der Prozessor*lokalen* Variable, die zweite mit dem Zugriff auf die der CPU zugeordneten Variable aus den einzelnen Programmfäden und die dritte mit der Initialisierung der globalen Datenstruktur.

1.2.1 Globale Definition

Die Definition der prozessorlokalen Variablen soll in der globalen Headerdatei `percpu_variables.h` erfolgen und sich so nah wie möglich an den üblichen Variablendefinitionen orientieren. Im Idealfall soll an den Variablendefinitionen nicht sichtbar sein, dass diese prozessorlokal sind:

```
/* Per CPU variables are defined in this header file */
int myint;
```

Diese Datei soll nicht direkt von den Benutzern eingebunden werden.

1.2.2 Benutzung

Um die prozessorlokalen Variablen benutzen zu können, soll eine Headerdatei `percpu.h` angeboten werden. Diese definiert eine Zugriffsfunktion `PER_CPU(var)` und bindet die Datei `percpu_variables.h` so ein, dass die dort definierten Variablen für jede CPU instanziiert werden können.
Der Zugriff auf eine prozessorlokale Variable soll durch die Benutzung von `PER_CPU(var)` geschehen:

```
PER_CPU(myint) = 7;
printf("myint: %d\n", PER_CPU(myint));
```

Wenn möglich sollte `PER_CPU(var)` komplett als Makro implementiert werden, auf gar keinen Fall darf hier jedoch ein Systemaufruf durchgeführt werden (siehe 1.1)

1.2.3 Initialisierung

Vor dem Starten der einzelnen CPUs muss für die prozessorlokalen Variablen eigener Speicher angefordert werden. Dies sollte in der schon implementierten Funktion `__boot_cpus()` geschehen.

1.3 Testen der Implementierung

Als Test bietet es sich an, eine prozessorlokale Variable `int psid;` zu definieren und dieser auf jeder CPU einmal die aktuelle Prozessornummer zuzuweisen. Danach soll die Testfunktion von Übungsaufgabe 1 um die Ausgabe dieser Variable erweitert werden. Der Test ist erfolgreich, wenn die Prozessornummer und der Inhalt der Variable bei jeder Ausgabe den gleichen Inhalt aufweisen.

Aufgaben:

→ Optimierung von `__get_cpu_id()` unter Verwendung des Stapelzeigers

→ Identifikation der API für prozessorlokale Variablen und deren globale Definition

→ Implementierung der prozessorlokalen Variablen

→ Entwickeln einer Demonstrationsanwendung (vgl. 1.3)

Hinweise:

- Variante 1: `mmap(2)` bietet die Möglichkeit, eine Adresse als Basisadresse vorzugeben. Diese wird vom Linux-Kernel übernommen, wenn sie an einer Seitengrenze ausgerichtet ist und an dieser Stelle noch kein anderer Speicher in den Adressraum des Prozesses eingeblendet ist. Über geschickte Auswahl der Adresse und einer Maske für den Stapelzeiger lässt sich so die Prozessornummer bestimmen.

  Variante 2: Die vom System bestimmten Basisadressen der jeweiligen Prozessorstapels werden in einem globalen Suchfeld abgelegt, dessen Index der jeweiligen Prozessornummer entspricht. Zur Bestimmung kann nun der Stapelzeiger einem Suchfeldeintrag zugeordnet werden, da die Basisadresse und die Größe der einzelnen Stapelbereiche bekannt ist.

  Es bietet sich an, beide Varianten zu implementieren, und manuell über `#ifdef` konfigurierbar zu machen, da die erste Variante zwar schneller, aber nicht so portabel wie die zweite Variante ist.

- Um prozessorlokale Variablen zu definieren bietet es sich an, diese in einer Struktur (z.B. `struct __percpu`) einzutragen und einen globalen Zeiger auf ein Feld dieser Strukturen zu definieren. Jeder Eintrag in diesem Feld entspricht einer CPU.

1.4 Abgabe: am 28.05.2009

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