5 Übungsaufgabe #5: Verwaltung von Betriebsmitteln

Das Ziel dieser Aufgabe ist die Implementierung von nicht-blockierenden Kommunikations- und Koordinationsprimitiven.

5.1 Verdrängungssperre

Implementiert eine Verdrängungssperre ähnlich zu der in der Vorlesung vorgestellten Schleusenvariante.
Stellt dafür z.B. vereinfachte Varianten der Funktionen `CS_ENTER()` und `CS_LEAVE()` bereit (Siehe Vorlesung 6-1). In `CS_ENTER()` wird eine CPU-lokale Verdrängungssperre gesetzt. Darüberhinaus bietet es sich an, eine zusätzliche CPU-lokale Zustandsvariable *needReschedule* zu verwenden, die angibt, ob ein Faden neu eingeplant werden soll. Neben der Aufhebung der Verdrängungssperre überprüft `CS_LEAVE()` *needReschedule* und wechselt gegegebenenfalls zum nächsten laufbereiten Faden.
Anstatt die Verdrängung direkt im `SLIH` des Zeitgebers auszulösen wird nur signalisiert, dass eine Verdrängung stattfinden soll. Am Ende der Unterbrechungsbehandlung muss sowohl *needReschedule* als auch die Verdrängungssperre beachtet werden. Wenn die Verdrängungssperre nicht gesetzt ist, darf an dieser Stelle sofort ein Fadenwechsel vollzogen werden.
Überlegt euch, an welcher Stelle in der unteren Teilaufgabe es notwendig ist, diese Art der Synchronisation anzuwenden und warum.

5.2 Warten auf ein vordefiniertes Signal

Ein Faden der Benutzerebene kann sich auf ein Ereignis der Kernebene, nämlich den Empfang eines Signals, synchronisieren. Der zur Synchronisation verwendete Algorithmus (Semaphore) bietet hierzu folgende Schnittstelle an.

- `signal()` wird aus dem `SLIH` des Signals heraus aufgerufen und soll die Anzahl der aufgetretenen Signale zählen.
- `wait()` wird vom Faden auf Benutzerebene aufgerufen und wartet, bis ein Signal empfangen wird. Sind vorher schon Signale angekommen, deren Empfang nocht nicht durch den Aufruf von `wait()` „konsumiert“ wurden, so soll nicht blockiert werden.

Wende hierzu den in der Tafelübung besprochenen Algorithmus zur nicht-blockierenden Synchronisation von Aktivitatsträgern an, um das Lost-Wakeup Problem zu lösen. Beachtet dabei, dass dieser immer nur *einen* Faden blockieren kann. Dies sollt ihr in der Implementierung sicherstellen, so dass ein zweiter Aufrufer einen Fehler signalisiert bekommt.
Der Faden soll passiv warten, indem er z.B. mit Hilfe einer `block()`- Funktion im Scheduler die Kontrolle abgibt, ohne in die Bereitliste eingetragen zu werden. Analog dazu muss es auch eine `wakeup()`-Funktion geben, die den Faden wieder in die Bereitliste einträgt und damit den Faden deblockiert. Geht davon aus, dass immer genug Fäden in der Bereitliste vorhanden sind und diese nie leerlaufen kann.

Hinweise:

- Um ein Doppelwort-CAS im 64-bit Modus zu realisieren benötigt man auf x86-64 den Maschinenbefehl `cmpxchg16b`. Leider lässt sich dieser nicht ohne weiteres durch Verwendung der *GCC Atomic Builtins* (`http://gcc.gnu.org/onlinedocs/gcc-4.1.1/gcc/Atomic-Builtins.html` einbauen. Dies funktioniert, wenn man den nicht standardkonformen Datentypen `__int128` verwendet:

```
#include <stdio.h>

int main()
{
    __int128 a = 1, b;
    b = __sync_val_compare_and_swap(&a, 1, 2);
    printf("Returned value = %d, New value = %d\n", (int)b, (int)a);
    return 0;
}
```

  Darüberhinaus muss das Programm mit `-mcx16` übersetzt werden, damit der `GCC` den Befehl kennt.

5.3 Abgabe: am 18.07.2012