3 Übungsaufgabe #3: Zweistufige Unterbrechungen

In dieser Aufgabe soll basierend auf der einfachen Unterbrechungsbehandlung aus Aufgabe 2 das zweistufige Behandlungskonzept bestehend aus *First-* und *Second-Level Interrupt Handlern* implementiert werden.

3.1 Unterbrechungsbehandlung

Die Behandlung von Unterbrechungen in *BST* soll mit einem zweistufigen Konzept bestehend aus je einem *First-* und *Second-Level Interrupt Handler* implementiert werden:

1. Der *First-Level Interrupt Handler (FLIH)* wird bei jeder auftretenden Unterbrechung ausgeführt und hat die Aufgabe, eine nachgelagerte Unterbrechungsbehandlung (*Second-Level Interrupt Handler (SLIH)*) in eine Warteschlange einzuhängen und die Unterbrechung zu quittieren. Nach der Rückkehr aus dem *FLIH* sollen alle in der Warteschlange vorhandenen *SLIH* ausgeführt werden.

2. Der *Second-Level Interrupt Handler (SLIH)* erledigt die eigentliche Arbeit der Unterbrechungsbehandlung und kann durch den *FLIH* unterbrochen werden.

In der Vorlesung wurden verschiedene Varianten der Unterbrechungsbehandlung vorgestellt, die unterschiedliche Anforderungen an die Synchronisation und Anzahl der *SLIH*-Warteschlange(n) stellen. Im Rahmen dieser Aufgabe sollen davon folgende Spielarten implementiert werden:

3.1.1 Behandlung von *FLIH* und *SLIH* durch denselben virtuellen Prozessor (für alle)

FLIH wie auch *SLIH* sollen auf der virtuellen CPU ausgeführt werden, der das auslösende Signal zugestellt wird. Dies entspricht der in der Vorlesung vorgestellten *Fließband*verarbeitung mit $x_v = x_n$.

3.1.2 Behandlung aller *SLIH*s auf einer vorgegebenen CPU (optional für 7.5 ECTS)

Alle *SLIH*s sollen auf einem zur Übersetzungszeit festgelegten Prozessor zur Ausführung gebracht werden. Die *FLIH*s hingegen sollen auf der durch das Signal unterbrochenen CPU ausgeführt werden. Dies entspricht der in der Vorlesung vorgestellten *Zusteller*variante.
Werden beide Varianten implementiert, so soll es möglich sein, eine konkrete Variante zum Übersetzungszeitpunkt über eine Präprozessorvariable auszuwählen. Die Implementierung erfolgt über `#ifdefs` im Programmcode, die den Code der gewünschten Variante auswählen.

Synchronisation

Die Propagation von *FLIH*s und *SLIH*s wird in beiden Fällen mit Hilfe einer Warteschlange implementiert. Jedoch treten dabei unterschiedliche Zugriffsmuster auf die Datenstrukturen auf. So bietet es sich an die in 3.1.1 beschriebene Variante derart zu implementieren, dass für jeden virtuellen Prozessor eine eigene *SLIH*-Warteschlange verwendet wird, und so alle Zugriffe auf die Warteschlange der *SLIH*s nur von einem Prozessor aus stattfinden. Werden, wie in 3.1.2, jedoch alle *SLIH*s auf einem Prozessor ausgeführt, so ist es notwendig, dass mehrere Prozessoren parallel auf dieselbe Warteschlange zugreifen.
Bei der Wahl der Synchronisationsprimitiven muss dem Rechnung getragen werden und je nach Situation korrekt synchronisiert werden.

Aufgaben:

→ Implementierung von *FLIH / SLIH* (mit Warteschlange) auf Basis der einfachen Unterbrechungsbehandlung mit Signalen aus Aufgabe 2.

→ Anforderungen an die Synchronisation der Warteschlange(n) bestimmen und umsetzen.

Hinweise:

- Synchronisation zwischen mehreren parallel arbeitenden Prozessoren ist mit Hilfe von Spinlocks erreichbar (vgl. Vorlesung Betriebssysteme)
- Auch für die Signalsynchronisation ist es sinnvoll diese hinter einer Schnittstelle zu kapseln, und somit dafür eine Abstraktionsschicht zu schaffen.
- Mit der GCC-Option `-DSymbolname` kann man beim Aufruf des Übersetzers die Präprozessorvariable `Symbolname` setzen.

3.2 Abgabe: am 22.06.2011