2 Übungsaufgabe #2: Aspektorientiere Signalverarbeitung & Konfigurierbarkeit

2.1 Aspektorientierte Signalverarbeitung

Nachdem ihr in der vorherigen Übung erste Erfahrungen mit der Programmiersprache AspectC++ gesammelt habt, sollt ihr nun damit im Rahmen dieser Aufgabe Teile eurer BST-Lösung mit Hilfe von Aspekten konfigurierbar gestallten.
Darüberhinaus wurde in der Vorlesung die Entwicklung von Software in Software-Produktlinien vorgestellt. Im zweiten Teil dieser Aufgabe sollt ihr eure bestehenden Implementierungsartefakte mit Hilfe von `Kconfig` als Software-Produktlinie darstellen und sie somit konfigurierbar machen.

2.1.1 Signalverteilung

In Aufgabe 2 wurden verschiedene Varianten der Signalverteilung für unterschiedliche Signale implementiert. Diese Implementierungen sollen nun mit Hilfe von Aspekten so umgebaut werden, dass man durch entsprechende Parametrierung eines Aspekts die Signalverteilung für jede Signalquelle beliebig einstellen kann.
Dabei bietet es sich an für jede (unterstützte) Signalnummer eine eigene Klasse zu erstellen, welche die Unterbrechungsbehandlung für das jeweilige Signal bereitstellen soll. Die Behandlungsfunktion sollte dann eine statische Methode dieser Klasse sein, um dem Interface für `sigaction` zu entsprechen. Die Signatur der statischen Funktion kann nun innerhalb eines *Pointcut*-Ausdrucks verwendet werden, um einen Aspekt an die Behandlung einer bestimmten Signalnummer zu binden.
Durch die Verwendung von *pure virtual Pointcuts* und Vererbung auf Aspekten kann nun erreicht werden, dass die Implementierung des Verteilungsmechanismus von der konkreten Signalquelle getrennt ist und durch Spezialisierung des Verteilungsaspektes ein beliebiges Signal als Quelle ausgewählt werden kann.

2.1.2 Signalbehandlung

Ähnliches soll für die in Aufgabe 3 implementierte zweistufige Interrupt-Behandlung realisiert werden. Die Implementierung der beiden Varianten soll nun in jeweils einen Aspekt refaktorisiert werden.
Abhängig davon, welcher der beiden Aspekte aktiviert ist, soll nun global im ganzen System die Abarbeitung der *SLIH*s auf der unterbrochenen virtuellen CPU, die auch den *FLIH* abgearbeitet hat, oder zentral auf einem vorher ausgewählten Prozessor stattfinden.
Dazu ist es sinnvoll SLIH und FLIH jeweils als statische Methode eines IRQ-Behandlungsobjektes zu implementieren und deren Ausführung als *Advice* zu implementieren.

Aufgaben:

→ Programmierung generischer Aspekte, um die Signalverteilung durch Aspektspezialisierung konfigurieren zu können.

→ Implementierung der beiden *FLIH*/*SLIH* Behandlungsvarianten mit Hilfe von Aspekten.

2.2 Merkmalmodell für die Übungsaufgabe

Die unterschiedlichen Lösungsvarianten, die ihr im Rahmen dieser Aufgabe implementiert habt, sollt ihr nun in ein Merkmalmodell (*feature model*) überführen.
Die Umsetzung erfolgt in der Sprache des Linux Konfigurationswerkzeugs `Kconfig`.

2.3 Das Linux Konfigurationswerkzeug: `make menuconfig`

Die Entwicklung mit Produktlinien sieht vor, dass auch nicht-Domänenexperten spezifische Lösungen generieren können. Für den Linux-Kern wurde dazu ein spezielles Werkzeug, `Kconfig`, entwickelt, welches es den Benutzern erlaubt, den Kern zu konfigurieren. Dieses Werkzeug wird in den Linux Quellen mit den Kommandos `make menuconfig` aufgerufen. `Kconfig` stellt sicher, dass die vom Benutzer getroffene Auswahl im Variantenmodell gültig ist. Im Fall des Linux-Kerns werden dann aus dieser Auswahl einbindbare *Header*-Dateien und *Make*-Fragmente erzeugt, die dann den Build-Prozess so beeinflussen, dass die ausgewählte Variante erzeugen wird.
Diese Werkzeugkette sollt ihr nun auch für eure Lösung nutzen. Allerdings sind die durch `Kconfig` erzeugten Header alleine noch nicht sehr nützlich, da ihr eure variablen Features nicht mit Hilfe von C-Präprozessordirektiven implementiert habt. Ein Feature wird stattdessen durch Hinzufügen von Aspekten implementiert. Die durch KConfig beschriebenen Features müssen also noch auf Dateimengen abgebildet werden, die beim Übersetzungsvorgang berücksichtigt werden.
Hierzu stellen wir euch das Werkzeug `BST-config`. Damit könnt ihr ein Feature aus der Kconfig-Beschreibung durch Verwendung eines Familienmodells auf eine Menge von Implementierungsdateien abbilden. Diese Dateimenge, die zu einer Auswahl von Features gehört, kann nun mit Hilfe von `BST-config` in einen eigenen Ordner kopiert werden, in welchem ihr dann die konkrete Variante eurer Software übersetzen könnt.
Die im Rahmen dieser Aufgabe erstellten Varianten (Abschnitt 2.1.1 und 2.1.2) sollen nun mit Hilfe von `Kconfig` und `BST-config` auswählbar gemacht werden.

Aufgaben:

→ Modelliert die soweit vorhandene und implementierte Variabilität in Form eines Variantenmodell als Merkmaldiagramm (am Besten auf Papier).

→ Implementiert dieses Modell mit dem Linux-Konfigurationsprogramm `Kconfig`.

→ Integriert das Linux-Konfigurationsprogramm in eure vorhandenen Makefiles.

→ Arbeitet euch in das zur Verfügung gestellte Konfigurationssystem `BST-config` ein, um mit Hilfe der mit `Kconfig` erzeugten Konfiguration ein Projektverzeichnis zu konstruieren, in dem nur die Aspekte zur Übersetzung verwendet werden, die für die selektierte Konfiguration relevant sind.

Hinweise:

- Eine bereits vorübersetzte ausführbare Version des Linux Konfigurationswerkzeugs Kconfig liegt auf den Laborrechnern unter `/proj/i4bst/tools/bst-kconfig`. Er wird wie folgt gestartet: `mconf features.fm` und erzeugt dann aus euerer Feature-Auswahl eine `.config` Datei.
- Ebenfalls unter `/proj/i4bst/tools/bst-kconfig` findet ihr das Werkzeug `transform.pl`. Mit diesem könnt ihr einzelne Features auf Dateimengen abbilden. Der Aufbau des Familienmodells, das die Abbildung von Features in Kconfig auf Dateien festlegt, und die Bedienung des Werkzeugs sind in `bst_transform.pdf` dokumentiert. Darüberhinaus sind dort auch zwei Beispiele zu finden, an denen ihr den Aufbau der Konfigurationsdateien und das Zusammenspiel mit Kconfig nachvollziehen könnt.

2.4 Abgabe: am 27.06.2012