AUFGABE 3: TRIPLE MODULAR REDUNDANCY

In dieser Aufgabe werden Sie Ihren in Aufgabe 2 implementierten Filteralgorithmus **dreifach redundant** ausführen, um Hardwarefehler zu maskieren und erste Erfahrungen mit TMR zu sammeln. In späteren Aufgaben werden wir die Effektivität Ihrer Implementierung mittels Fehlerinjektion testen.

Die Vorgabe befindet sich im Ordner `03_tmr` des Vorgaben-Repositories:

```
git@gitlab.cs.fau.de:ezs/vezs19-vorgabe.git
```

Starten Sie die Anwendung mit `make run` im Build-Verzeichnis, nachdem Sie mittels

```
source ./ecosenv.sh && mkdir build && cd build && cmake ..
```

dieses erstellt haben.

Ziel der Aufgabe ist es ein typisches redundantes Filtersystem zu entwerfen. Dafür stellen wir Ihnen drei virtuelle Sensoren zu Verfügung, die (redundant) jeweils dieselbe Messgröße erfassen.

1 Aufgabenstellung

1.1 Konzeption

Aufgabe 1 *Fehlerhypothese*

Bevor Sie anfangen die Filterimplementierung gegen transiente Fehler zu schützen, benötigen Sie eine Fehlerhypothese. Gehen Sie für diese Aufgabe von einer trivialen, *nicht redundanten* Signalverarbeitungskette aus: Sensorik → Verarbeitung → Aktorik. *An welchen Stellen können Fehler auftreten, wenn man Speicherfehler/Registerfehler annimmt? Welche Folgen können diese Fehler beispielsweise haben?* Zählen Sie mindestens 3 Folgen auf.

Antwort:

Aufgabe 2 *Fehlerbaum*

Analysieren Sie die Filterimplementierung mit Hilfe der *Fehlerbaummethode* und halten Sie das Analyseergebnis schriftlich fest.

Antwort:

Aufgabe 3 *Grundstruktur*

Erstellen Sie ein Taskgerüst — zunächst ohne konkrete Implementierung — für die periodische Ausführung Ihrer Signalverarbeitungskette. Legen Sie für alle benötigten Schritte (*Eingabedaten erfassen*, *Filtern*, *Daten ausgeben*) eigene eCos-Fäden[1] an. *Worauf müssen Sie bei der Implementierung der Rangfolge zwischen den einzelnen Fäden achten, wenn man annimmt, dass Fehler bei der Ausführung der einzelnen Fäden auftreten können?* Konsultieren Sie bei Fragen bitte zunächst die eCos-Dokumentation[2] und erst wenn Sie nicht mehr weiter wissen die Rechnerübungsleiter.

☞ cyg_flag_timed_wait()

Antwort:

1.2 Implementierung

Implementieren Sie die Signalverarbeitung mit einer Periode von 2 ms. Der auslösende Alarm wurde schon von uns angelegt.

Aufgabe 4 *Sensorwerte*

Fragen Sie die Messgröße des redundant ausgelegten Sensorsystems ab. Wie bei analogen Sensoren üblich geben die Sensoren eine positive Ganzzahl vom Typ

☞ ezs_getValueSensorA()
☞ ezs_getValueSensorB()
☞ ezs_getValueSensorC()

[1] http://ecos.sourceware.org/docs-latest/ref/kernel-thread-create.html
[2] http://ecos.sourceware.org/docs-latest/ref/ecos-ref.html

`uint16_t` zurück. Diese Zahl repräsentiert eine Spannung zwischen 0V (`0U`) und 15V (`UINT16_MAX`). Berechnen Sie die anliegende Spannung in Volt und rechnen Sie mit dieser Zahl in Q-Notation weiter.

Aufgabe 5 *Replizierte Filterung*

Filtern Sie die abgefragten Sensorwerte nach den in Vorlesung und Übung vorgestellten Prinzipien **dreifach redundant**. Achten Sie besonders auf den Replikdeterminismus und eine saubere Kapselung der einzelnen Replikate. Nutzen Sie wie in Teilaufgabe 3 vorgesehen einen **eigenen Aktivitätsträger** für jedes Replikat. Kopieren Sie für die Filterung selbst Ihre Filterimplementierung aus Aufgabenblatt 2 an die vorgesehenen Stellen in den Dateien `src/filter.c` und `include/filter.h`. Behalten Sie unsere Schnittstellen bei und passen Sie Ihren Code entsprechend an, falls notwendig.

Aufgabe 6 *Ausgangsmaskierung*

Führen Sie nun die Ergebnisse der Filterung in einem Ausgangsvoter zusammen und vergleichen Sie die Ergebnisse der Filterung. Beenden Sie die Ausführung Ihrer Anwendung falls Sie einen Zustand feststellen, von dem sich das System nicht mehr erholen kann. Achten Sie darauf, dass Sie keine Werte aus der alten Runde verwenden.

Normalerweise wäre die Ausnahmebehandlung für einen wiederherstellbaren Fehler ebendiese Wiederherstellung. Da wir im aktuellen, emulierten System aber mit an Sicherheit grenzender Wahrscheinlichkeit keine Hardwarefehler erfahren werden, beenden Sie Ihr System auch im Falle eines wiederherstellbaren Fehlers, da dies vermutlich eine Verletzung des Replikationsdeterminsmus bedeutet. Das heißt, auch die Abwesenheit von Einstimmigkeit muss als Fehler gewertet werden. *Um welche Art von Test handelt es sich bei Ihrem Voter?* Was müsste *normalerweise* beim Erkennen eines Fehlers in *einem* Replikat getan werden?

Antwort:

Geben Sie den gefilterten Wert „aus“ indem Sie `ezs_setOutputA()` aufrufen.

1.3 Evaluation

Aufgabe 7 *Replikdeterminismus*

Führen Sie das System für eine Minute ohne Fehler aus. Falls Sie Fehler entdecken, spricht dies vermutlich für eine Verletzung des Replikdeterminismus. Finden Sie etwaige solche Verstöße und eliminieren Sie diese.

Aufgabe 8 *TMR Analyse (I)*

Welche neuen möglichen Angriffsflächen für transienten Fehler sind durch die Implementierung von Triple Modular Redundancy hinzugekommen?

Antwort:

Aufgabe 9 *TMR Analyse (II)*

Wie kann die tatsächliche Effektivität Ihrer TMR-Maßnahmen überprüft werden?

Antwort:

Aufgabe 10 *Sphere of Replication*

Bestimmen Sie die durch Redundanz gesicherten Bereiche in Ihrem Programm (→ Sphere of Replication) und markieren Sie diese durch Kommentare in Ihrem Quellcode.

2 Erweiterte Aufgaben

Aufgabe 11 *Analyse*

Untersuchen Sie, wie in der Tafelübung vorgestellt, Ihr Kompilat. Welche Teile sind tatsächlich repliziert?

☞ objdump

Antwort:

Aufgabe 12

Wo befinden sich noch Lücken in der Replikation? Wie können diese geschlossen werden?

Antwort:

Aufgabe 13 *Umsetzung*

Nutzen Sie die Datei `include/filter.hpp` um mithilfe der in der Tafelübung gezeigten Programmiermaßnahmen eine Codeduplizierung zu erreichen. Übernehmen Sie Ihre bisherige Änderungen in die Datei `src/app_ext.cpp`. Überschreiben Sie die Datei nicht: Sie ist schon für die Verwendung von C++ angepasst. Mit `make tmr_ext` und `make run_tmr_ext` können Sie die neue Implementierung kompilieren und testen.

☞ C++-Templates

Hinweise

- Bearbeitung: Gruppenarbeit
- Abgabefrist: 04.06.2019
- Fragen bitte an `i4ezs@lists.cs.fau.de`