AUFGABE 2: DREIFACH REDUNDANTE AUSFÜHRUNG

In dieser Aufgabe implementieren Sie TMR-Varianten für einen gegebenen Filteralgorithmus. Verwenden Sie zur Quellcodeverwaltung `git`. Die tatsächliche Funktionalität des Filters ist für die Bearbeitung der Aufgabe nicht relevant. Achten Sie nur auf die korrekte Verwendung (Aufruf, Eingabe/Ausgabewerte).

Die Effektivität Ihrer Implementierung wird mittels eines Fehlerinjektionsexperiments getestet. In der Vorgabe finden Sie eine vollständige, allerdings noch ungesicherte, Ausführung des Algorithmus.

Achten Sie bitte auf die Kommentare in den Vorgaben, welche Stellen angefasst werden sollen/dürfen und welche unangetastet bleiben sollen.

Legen Sie die vorherige Aufgabe in den Unterordner `aufgabe1` und übertragen Sie die Änderung in Ihr git-Repository. Laden Sie die aktuelle Vorgabe mittels git herunter.

Altes Vorgabe-Repo entfernen:

```
git remote remove vorgabe
```

Neues Vorgabe-Repo einbinden:

```
git remote add vorgabe gitosis@i4git:vezs_ss14_vorgabe
```

Vorgabe herunterladen:

```
git pull vorgabe master
```

Umgebungsvariablen setzen:

```
cd aufgabe2; source ecosenv.sh
```

Aufgabenstellung

1. *Ungesichertes System:* Kompilieren Sie die Vorgabe (in `app_plain.c`). Sichern Sie die Ergebnisse unter dem Variantennamen *plain*:
`ccmake .` → `FAILVARIANT`
Führen Sie die Injektion, wie in der Übung vorgestellt durch (`make fail-1-trace, make fail-2-import, etc.`) und betrachten Sie die Ergebnisse: `make fail-5-browse`.
An welchen Stellen entstehen die meisten Fehler? Versuchen Sie anhand des Dumps (`tt_scope.dis`) die entsprechenden Stellen im C-Code zu identifizieren.

2. *TMR-Ausführung:* Entwerfen Sie eine dreifach redundante Ausführung des Filteralgorithmus (in `app_tmr.c`). Ändern sie entsprechend `app_plain.c` in `CMakeLists.txt` in `app_tmr.c` um, um die korrekte Datei zu übersetzen. Behalten Sie die Schnittstelle von `convolve()` bei, Rückgabewert und Parameter von `process()` können Sie nach belieben anpassen!
Replizieren Sie die Eingabedaten auf geeignete Weise und entwerfen Sie einen Votingalgorithmus `vote()` zum Vergleichen der Ergebnisse. Auch hier haben Sie freie Hand bei Parameter- oder Rückgabewahl.
Benennen Sie die Variante als *TMR_base* (`ccmake .` → `FAILVARIANT`) und führen Sie eine erneute Fehlerinjektion durch (`make fail-1-trace`, `make fail-2-import`, etc.)
Konnten Sie die *NEGATIVE MARKER* verringern? An welchen Stellen finden sich noch unbemerkte Fehler?

3. *SoR:* Bestimmen Sie die durch Redundanz gesicherten Bereiche in ihrem Programm (→ Sphere of Redundancy) und markieren Sie diese durch Kommentare in ihrem Quellcode.
Können eventuell noch ungeschützte Stellen in ihrer Implementierung in den Redundanzbereich aufgenommen werden? Versuchen Sie *Silent Data Corruptions* weiter zu minimieren.
Implementieren Sie ihre optimierte Variante in `app_tmr_opt.c` und verwenden Sie den Fail*-Variantennamen *TMR_opt* für die Fehlerinjektion.

Die Fehlerinjektion kann durchaus einige Zeit in Anspruch nehmen. Das ungesicherte System benötigt auf acht Kernen (z.B. faui00a) ca. 10 Minuten. Nutzen Sie die Zeit, um sich bereits Gedanken über mögliche Verbesserungen zu machen.

Hinweise

- Erforderliche Dateien: keine (git)
- Bearbeitung: Gruppe mit je zwei bis drei Teilnehmern.
- Abgabezeit: 20.05.2013
- Fragen bitte an `i4ezs@lists.informatik.uni-erlangen.de`