Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

Florian Franzmann Tobias Klaus

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
https://www4.cs.fau.de

6. Juli 2015

Überblick

1 Überblick

2 C-Quiz Teil VI

3 Wiederholung Software-TMR

4 Eliminierung von Bruchstellen in TMR

5 Aufgabenstellung

Überblick

1 Überblick

2 C-Quiz Teil VI

3 Wiederholung Software-TMR

4 Eliminierung von Bruchstellen in TMR

5 Aufgabenstellung

Annahmen

- C99
- x86 bzw. x86-64, d. h.
  - vorzeichenbehaftete Integer als Zweierkomplement implementiert
  - `char` hat 8 Bit
  - `short` hat 16 Bit
  - `int` hat 32 Bit
  - `long` hat 32 Bit auf x86 und 64 Bit auf x86-64

Frage 16

Angenommen `x` hat Typ `int`. Ist `x + 1` …

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von `x`?

Frage 16

Angenommen `x` hat Typ `int`. Ist `x + 1 ...`

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von `x`?

Erklärung

- nicht definiert genau dann, wenn `INT_MAX`

Frage 17

Angenommen `x` hat Typ `int`. Ist `x - 1 + 1` ...

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von `x`?

Frage 17

Angenommen `x` hat Typ `int`. Ist `x - 1 + 1` ...

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von `x`?

Erklärung

- additive Operatoren sind linksassoziativ

$\Rightarrow$ nicht definiert für `INT_MIN`

Frage 18

Angenommen `x` hat Typ `int`. Ist `(short)x + 1` ...

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von `x`?

Frage 18

Angenommen `x` hat Typ `int`. Ist `(short)x + 1` ...

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von `x`?

Erklärung

- wenn `x` nicht in `short` passt
  - $\rightsquigarrow$ implementierungsabhängig

Überblick

1 Überblick

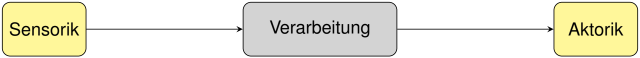
2 C-Quiz Teil VI

3 Wiederholung Software-TMR

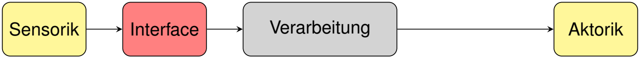
4 Eliminierung von Bruchstellen in TMR

5 Aufgabenstellung

Klassische “Triple Modular Redundancy” (TMR)

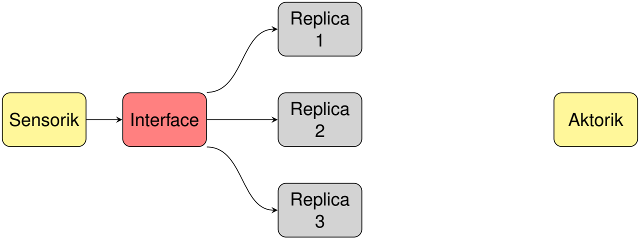


Klassische “Triple Modular Redundancy” (TMR)



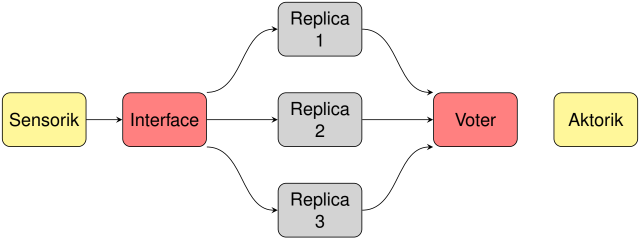
- Schnittstelle sammelt Eingangsdaten (Replikdeterminismus)

Klassische “Triple Modular Redundancy” (TMR)



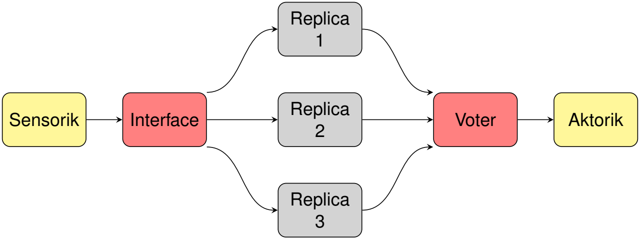
- Schnittstelle sammelt Eingangsdaten (Replikdeterminismus)
- Verteilt Daten und aktiviert Replikate

Klassische “Triple Modular Redundancy” (TMR)



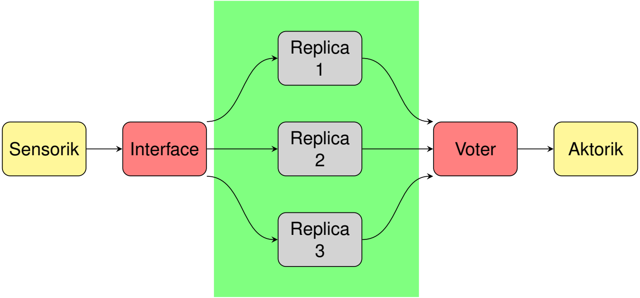
- Schnittstelle sammelt Eingangsdaten (Replikdeterminismus)
- Verteilt Daten und aktiviert Replikate
- Mehrheitsentscheider (Voter) wählt Ergebnis

Klassische “Triple Modular Redundancy” (TMR)



- Schnittstelle sammelt Eingangsdaten (Replikdeterminismus)
- Verteilt Daten und aktiviert Replikate
- Mehrheitsentscheider (Voter) wählt Ergebnis
- Ergebnis wird an Aktuator versendet

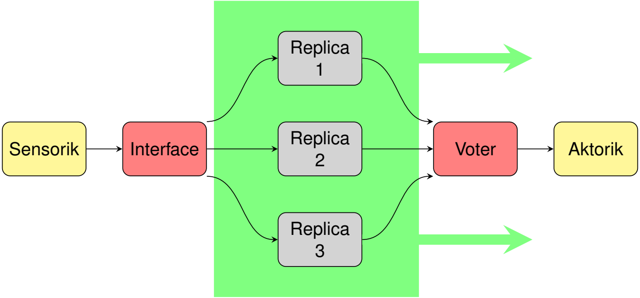
Klassische “Triple Modular Redundancy” (TMR)



Redundanzbereich

Ausschließlich Replikatausführung.

Klassische “Triple Modular Redundancy” (TMR)



Redundanzbereich

Ausschließlich Replikatausführung.

- ⇝ Erweiterung der Ausgangsseite mit Informationsredundanz
- ⇝ Mehrheitsentscheid über codierte Prüfsumme

Überblick

1 Überblick

2 C-Quiz Teil VI

3 Wiederholung Software-TMR

4 Eliminierung von Bruchstellen in TMR

5 Aufgabenstellung

Erweiterte arithmetische Codierung

nach Forin 1989: “Vital coded microprocessor principles and application for various transit systems” [1]

- Arithmetisch codierter Wert V_C
- Ausgangswert

$$V_C = \boxed{V}$$

Erweiterte arithmetische Codierung

nach Forin 1989: “Vital coded microprocessor principles and application for various transit systems” [1]

- Arithmetisch codierter Wert V_C
- Ausgangswert

$$V_C = \boxed{V} * \boxed{A}$$

Bitfehlererkennung (Restfehlerwahrscheinlichkeit $P = 1/A$)

- Schlüssel

Erweiterte arithmetische Codierung

nach Forin 1989: “Vital coded microprocessor principles and application for various transit systems” [1]

- Arithmetisch codierter Wert V_C
- Ausgangswert

$$V_C = \boxed{V} * \boxed{A} + \boxed{B_V}$$

Adressierungsfehlererkennung

- Schlüssel
- Variablenspezifische Signatur

Erweiterte arithmetische Codierung

nach Forin 1989: “Vital coded microprocessor principles and application for various transit systems” [1]

- Arithmetisch codierter Wert V_C
- Ausgangswert

$$V_C = \boxed{V} * \boxed{A} + \boxed{B_V} + \boxed{D}$$

Erkennung veralteter Daten

- Schlüssel
- Variablenspezifische Signatur
- Zeitstempel

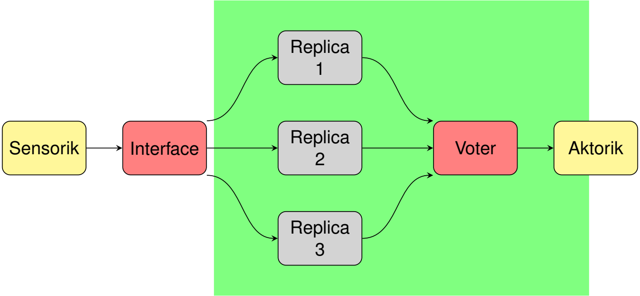
Wertebereichseinschränkungen

- Schlüssel A sollte so groß wie möglich sein:

$\leadsto$ Möglichst geringe Restfehlerwahrscheinlichkeit ($P = 1/A$)

- Wertebereich des dynamischen Zeitstempels
  - $D = \{x | x \in \mathbb{N}_0 \land x \leq D_{max}\}$
  - Zeitstempel darf überlaufen: $D_{max} + 1 = 0$
- Für jede Signatur B_* muss dann gelten
  - $B_* + D_{max} < A$
  - Die minimale Distanz zwischen jeweils zwei Signaturen im System muss kleiner D_{max} sein: $\forall i, j : |B_i - B_j| < D_{max}$

Erweiterung I – codierte Ausgangswerte



- Replikate liefern arithmetisch codierte Ergebnisse
- Mehrheitsentscheid auf codierten Prüfsummen
- Übertragung codierter Ergebnisse

Vereinfachung für diese Übung

Für diese Übungsaufgabe:

- Keine Datendiversität am Eingang
- Kein Zeitstempel
- Nur Absicherung der Ausgangsseite!

EAN Vergleichsoperator

- Voting basiert auf codierter Vergleichsoperation:

$\leadsto$ $X_C = Y_C \Rightarrow X * A + B_X = Y * A + B_Y$

- Im fehlerfreien Fall gilt:

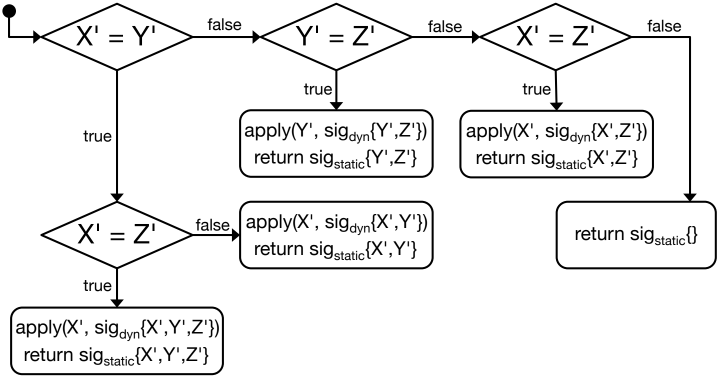
$X = Y$, $A = A$ aber $B_X \neq B_Y$!

  - *Rohwerte* sind identisch
  - Schlüssel ist per Definition identisch
  - Signaturen sind unterschiedlich (aber konstant!)

Bestimmung der Gleichheit durch Differenzbildung:

$\leadsto X_C - Y_C = B_X - B_Y = const.$

Codierter Mehrheitsentscheid



- Bestimmung von dynamischer und statischer Signatur:

$\rightsquigarrow$ $sig_{dyn}(X', Y') : X' = Y' \Rightarrow X' - Y'$

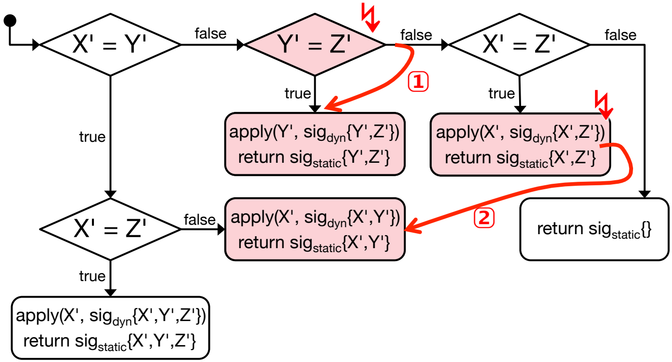
$\rightsquigarrow$ $sig_{static}(X', Y') : X' = Y' \Rightarrow B_X - B_Y$

Codierter Mehrheitsentscheid (Forts.)

1. Vergleichsoperation wird durchgeführt (z. B. $X' = Y' \wedge X' = Z'$)
   - Berechnung von sig_{dyn}
   - Vergleich mit sig_{static}
2. Verzweigunsentscheidung wird nachberechnet:
   - Wiederholte (redundante) Berechnung von sig_{dyn}
   - Addiere sig_{dyn} (*apply*) zum gewählten Ergebnis
3. Konstante Signatur des durchlaufenen Zweiges identifiziert Gewinner (Rückgabewert: sig_{static})
   - Aktor wählt entsprechendes Replikatergebnisse
   - führt inverse Operation zu *apply* durch

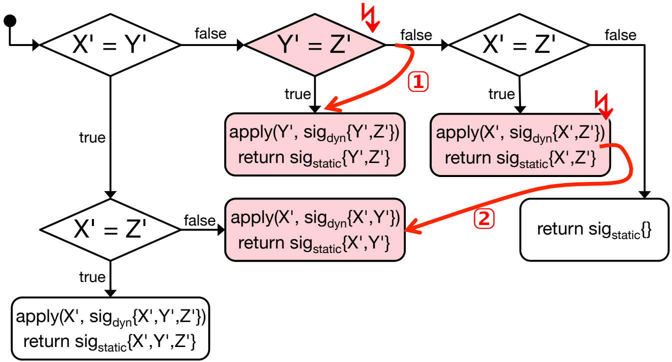
Im Voter wurde die *dynamisch berechnete Signatur der Verzweigungsentscheidung* hinzuaddiert. Im Aktor wird mit der entsprechenden *konstanten Signatur zurückgerechnet*.

Codierter Mehrheitsentscheid - Fehlerfall



1. Falsche Verzweigungsentscheidung: ($Y' \neq Z'$)
   - Y' wird als korrekt angenommen, sig_{dyn} wird berechnet
   - allerdings ist sig_{dyn} tatsächlich $\neq sig_{static}$
   - Fehler wird bei der inversen Operation zu *apply* erkannt

Codierter Mehrheitsentscheid - Fehlerfall



2. Falscher (plötzlicher) Sprung
   - X' wird als korrekt erkannt, sig_{dyn} wird erneut berechnet
   - Ein fehlerhafter Sprung zu einem anderen Block führt zu einem inkonsistenten Rückgabewert $sig_{static}\{X', Z'\}$
   - $sig_{dyn}\{X', Y'\} \neq sig_{static}\{X', Z'\}$ wird beim decodieren erkannt

Überblick

1 Überblick

2 C-Quiz Teil VI

3 Wiederholung Software-TMR

4 Eliminierung von Bruchstellen in TMR

5 Aufgabenstellung

Aufgabenstellung

Aufgabe

- Erweitern Sie Ihre Filterimplementierung um TMR
- Sicheren Sie den Voter per EAN ab

- Jedes Replikat hat genau einen Ausgabewert (integer $\mapsto$ enc_t): Eine kodierte Prüfsumme des Ergebnisses
  - $\leadsto$ Legen Sie für jede der drei Ausgabewerte (X', Y', Z') jeweils *unterschiedliche* aber *konstante* Signaturen (SIG_X, SIG_Y, SIG_Z) fest
- Nutzen Sie für X' den nächstgrößeren Datentyp zu X
  - $\leadsto$ Wählen Sie eine Zahl A mit möglichst großem Hamming-Abstand, *vermeiden Sie* dabei mögliche *Überläufe bei der Codierung*

Hinweise

- In dieser Aufgabe betrachten wir nur die Ausgangsseite
- Die Eingangsseite bleibt vorerst „ungeschützt“
- die besten Kandidaten findet ihr hier:
  `www4.cs.fau.de/Research/CoRed/experiments`
  - 8 Bit-Schlüssel: 185 und 233 ($d_h = 4$)
  - 16 Bit-Schlüssel: 58659, 59665, 63157, 63859 und 63877 ($d_h = 6$)

Für jede Operation zwischen zwei codierten Werten

ist eine eigene Funktion mit konstanten Signaturwerten notwendig!

Fragen?