Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

Florian Franzmann, Martin Hoffmann, Tobias Klaus

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

4. Juni 2014

Überblick

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 7

Zu was wird `INT_MAX + 1` ausgewertet?

1. `0`
2. `1`
3. `INT_MAX`
4. `UINT_MAX`
5. nicht definiert

Erklärung

`signed int`-Überlauf ist nicht definiert.

Frage 8

Zu was wird `-INT_MIN` ausgewertet?

1. `0`
2. `1`
3. `INT_MAX`
4. `UINT_MAX`
5. `INT_MIN`
6. nicht definiert

Erklärung

Es gibt keine Zweierkomplementdarstellung für `-INT_MIN`

Frage 9

Angenommen `x` hat Typ `int`. Ist `x << 0` ...

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

von `x`?

Erklärung

- negative Werte können nicht nach links verschoben werden
- noch nicht einmal um `0` Bit

Überblick

Ziele des Softwareentwurfs

Modifizierbarkeit: lokale Veränderbarkeit

- ⇝ Änderungen an Anforderungen umsetzbar
- ⇝ Fehler korrigierbar

Effizienz: optimaler Betriebsmittelbedarf

- wird häufig zu früh berücksichtigt

Verlässlichkeit: über lange Zeit Funktionsfähigkeit ohne menschlichen Eingriff

- gutmütiges Ausfallverhalten
- muss von Anfang an eingeplant sein!

Verständlichkeit: Isolierung von

- Daten
- Algorithmen

Prinzipien des Softwareentwurfs

Abstraktion: wichtige Details hervorheben

Kapselung: unnötige Details verbergen

Einheitlichkeit: konsistente Notation

Vollständigkeit: alle wichtigen Aspekte berücksichtigt

Testbarkeit: muss von Anfang an eingeplant werden

C macht es einem hier nicht leicht

$\rightsquigarrow$ disziplinierte Herangehensweise notwendig!

Fragestellung

Wie komme ich von der Beschreibung zur Software?

Objektorientierter/Objektbasierter Entwurf [1]

1. identifiziere Objekte und deren Attribute
2. identifiziere Operationen jedes Objekts
3. lege Sichtbarkeit fest
4. lege Objektschnittstellen fest
5. implementiere Objekte

Beispielbeschreibung – α-Filter

- $\hat{x}[\kappa]$ Schätzung, α Filterparameter, $y[\kappa]$ Messwert
- Initialisierung: $\hat{x}[0] = 0$
- Filterschritt:

$$r[\kappa] = y[\kappa] - \hat{x}[\kappa - 1] \quad (1)$$
$$\hat{x}[\kappa] = \hat{x}[\kappa - 1] + \alpha \cdot r[\kappa] \quad (2)$$

- Optimale Parameter (σ_w^2 Prozessvarianz, T Abtastintervall, σ_v^2 Rauschvarianz):

$$\lambda = \sigma_w \cdot T^2 / \sigma_v \quad (3)$$
$$\alpha = \left(-\lambda^2 + \sqrt{\lambda^4 + 16\lambda^2}\right)/8 \quad (4)$$

1. Objekte und Attribute identifizieren

- Herangehensweise:
  - Hauptwortextraktion aus Anforderungsdokument
  - für kleinere Probleme: *Intuition*
- Was ist das Objekt? $\rightsquigarrow$ Filter
- Attribute? Welche Information brauche ich für jeden Filterschritt?
  - Schätzung aus der Vorrunde $\hat{x}[\kappa - 1]$
  - Filterparameter α
  - aktuellen Messwert $y[\kappa]$ $\rightsquigarrow$ kein Zustand, kommt von aussen

Vorläufige Objektschablone

```
1 typedef struct _Alpha_Filter {
2   AF_Value_t x;
3   AF_Value_t alpha;
4 } Alpha_Filter;
```

2. Operationen identifizieren

- Herangehensweise:
  - Verbenextraktion
  - für kleinere Probleme: *Intuition*
- Leben eines Objekts:
  1. Initialisierung ⇝ Betriebsmittel anfordern
  2. Verwendung
  3. Beseitigung ⇝ Betriebsmittel freigeben
- Was möchten Benutzer mit dem Filter machen?
  - Filter initialisieren
  - Filterschritt ausführen
  - Schätzwert erfragen
  - Betriebsmittelfreigabe nicht notwendig

3. Sichtbarkeit festlegen

- in modernen Programmiersprachen `private`, `public`, ...
- in C nur eingeschränkt möglich
  - modulintern vs. modulextern
- Leitfaden: möglichst wenig sichtbar machen
  ⇝ öffentliche Schnittstelle bedeutet Verpflichtung
- Was soll bei unserem Filter öffentlich sein?
  - Initialisierung
  - Filterschritt
  - Schätzung abfragen
- alle anderen Operationen modulintern
  ⇝ Hilfsfunktionen `static`

4. Schnittstelle festlegen

- zwischen Modul und Außenwelt
- statische Semantik

Schnittstelle

```
1  void afilter_init(Alpha_Filter *filter,
2                    AF_Value_t process_variance,
3                    AF_Value_t noise_variance,
4                    AF_Value_t sampling_interval);
5
6  void afilter_step(Alpha_Filter *filter,
7                    AF_Value_t measurement);
8
9  AF_Value_t afilter_get_estimate(Alpha_Filter *filter);
```

5. Implementierung – Header

alpha_filter.h

```
1  #ifndef ALPHA_FILTER_H_INCLUDED
2  #define ALPHA_FILTER_H_INCLUDED
3
4  typedef float AF_Value_t;
5  typedef struct _Alpha_Filter {
6    AF_Value_t x;
7    AF_Value_t alpha;
8  } Alpha_Filter;
9
10 void afilter_init(Alpha_Filter *filter,
11                   AF_Value_t process_variance,
12                   AF_Value_t noise_variance,
13                   AF_Value_t sampling_interval);
14
15 void afilter_step(Alpha_Filter *filter,
16                   AF_Value_t measurement);
17
18 AF_Value_t afilter_get_estimate(Alpha_Filter *filter);
19 #endif // ALPHA_FILTER_H_INCLUDED
```

5. Implementierung – Initialisierung

$$\hat{x}[0] = 0 \tag{5}$$

$$\lambda = \sigma_w \cdot T^2/\sigma_v \tag{6}$$

$$\alpha = \left(-\lambda^2 + \sqrt{\lambda^4 + 16\lambda^2}\right)/8 \tag{7}$$

alpha_filter.c

```
1  void afilter_init(Alpha_Filter *filter,
2                    AF_Value_t process_variance,
3                    AF_Value_t noise_variance,
4                    AF_Value_t sampling_interval) {
5    filter->x = 0;
6    AF_Value_t l = sqrt(process_variance)
7      * sampling_interval * sampling_interval
8      / sqrt(noise_variance);
9    filter->alpha = (-l*l
10     + sqrtf(l*l*l*l + 16.0f*l*l)) / 8.0f; }
```

5. Implementierung – Filterschritt

$$r[\kappa] = y[\kappa] - \hat{x}[\kappa - 1] \tag{8}$$

$$\hat{x}[\kappa] = \hat{x}[\kappa - 1] + \alpha \cdot r[\kappa] \tag{9}$$

alpha_filter.c

```
1  void afilter_step(Alpha_Filter *filter,
2                    AF_Value_t measurement) {
3    AF_Value_t  r = measurement - filter->x;
4    filter->x = filter->x + filter->alpha * r;
5  }
6
7  AF_Value_t afilter_get_estimate(Alpha_Filter *filter)
8  {
9    return filter->x;
10 }
```

Literatur

Grady Booch.
Software Engineering with Ada.
The Benjamin/Cummings Publishing Company, Inc., 2nd edition, 1987.

Fragen?