Spoľahlivosť a kontrola toku programu

Vnorené systémy

Maroš Ďuríček

Ústav počítačových systémov a sietí
FIIT, STU BA

8.10.2014

Kontakt: maros.duricek@stuba.sk

STU FIIT

Obsah

STU FIIT

Úvod

Príčiny chýb vo vnorených systémoch:
(Chyba - prejavenie poruchy)

- fyzické poruchy - vnútorné a vonkajšie (žiarenie)
- poruchy v dôsledku ľudskej chyby

Prejavenie porúch:

- bez prejavenia
- zlyhanie
- detekcia poruchy - softvérovo, hardvérovo, timeout
- latentná porucha
- opravenie poruchy

Chyby a modely

Chyby v systémoch:

- hardvérová úroveň - preklopenie bitu . . .
- systémová úroveň
  - chyby v dátach
  - chyby v programe
    - Typ 1 - nezmení sa tok programu
    - **Typ 2 - zmení sa tok programu**
- softvérové chyby. . . chyby programátorov

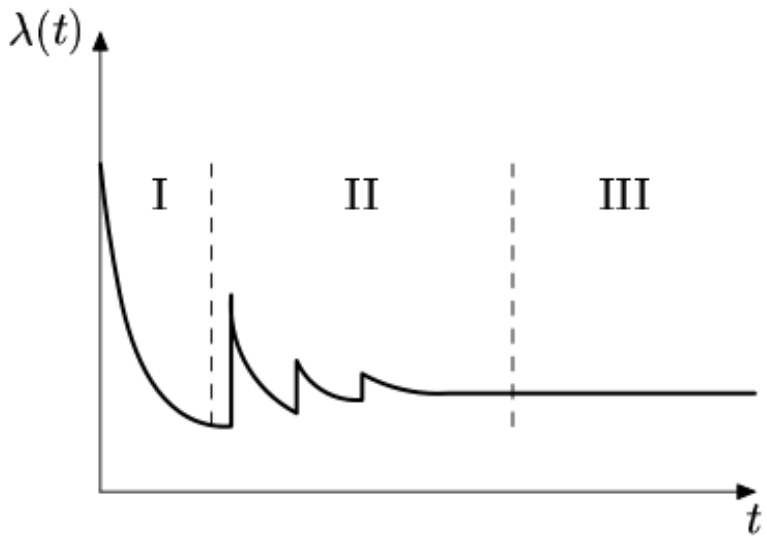
STU FIIT

Spoľahlivosť

Intenzita hardvérových porúch λ v závislosti od času:

$$\lambda = \pi_L \pi_Q \left(C_1 \pi_T \pi_V + C_2 \pi_E \right)$$

„Vaňová krivka“ softvéru - vsuvka



I - testovanie/ladenie - odstraňovanie chýb
II - používanie SW - modernizácia SW (upgrade) - vznik ďalších chýb
III - starnutie SW - λ ostáva konštantná

STU FIIT

Spoľahlivosť (2)

Dependability = „ability of a processor-based system to deliver a service that can justifiably be trusted“

Spoľahlivosť (*Dependability*) zahŕňa:

- Dostupnosť (*Availability*)
- *Reliability* - „pravdepodobnosť bezporuchovej prevádzky“
- Bezpečnosť (*Safety*) - schopnosť odvrátiť katastrofické následky pre ľudí/okolie
- Integrita
- Udržiavateľnosť (*Maintainability*)

Niektoré atribúty spoľahlivosti

$$R(t) = exp\left(-\int_0^t \lambda(\tau)\, d\tau\right)$$

Ak λ je konštantná, potom:

$$R(t) = e^{-\lambda t}$$

$$Q(t) = 1 - R(t)$$

$$f(t) = \lambda R(t) \text{ (hustota porúch)}$$

$$MTTF = \int_0^\infty e^{-\lambda t} dt = \frac{1}{\lambda}$$

Reliability - R(t)

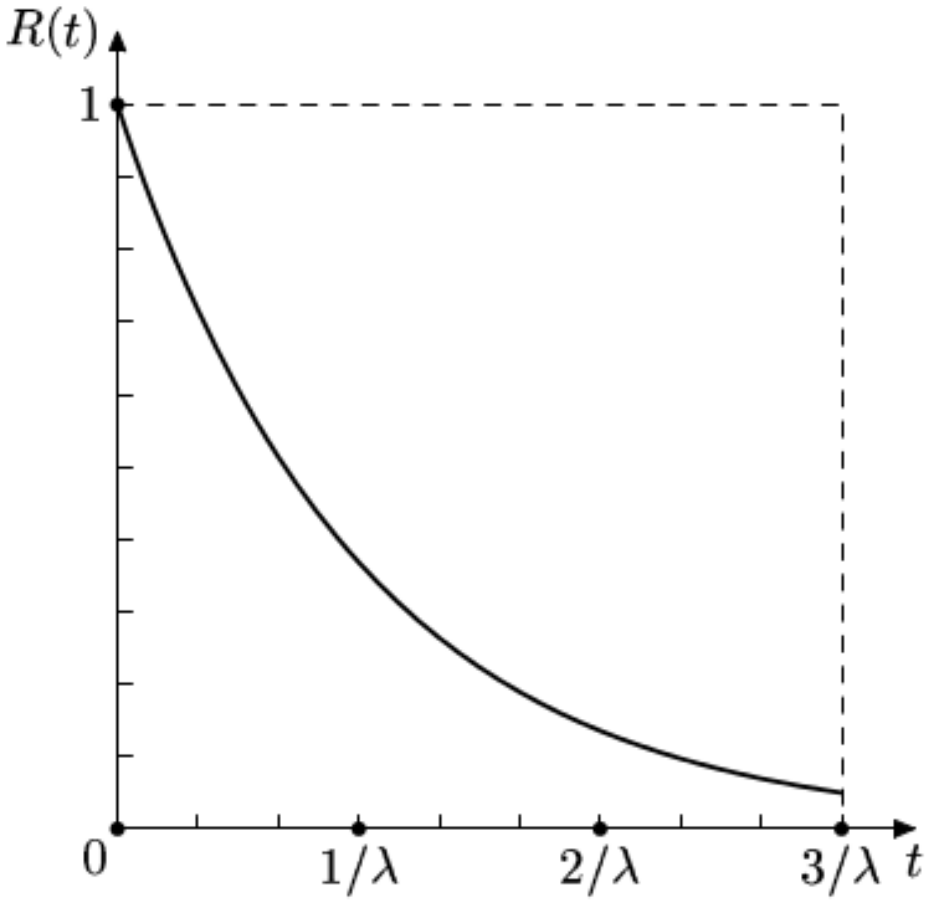


Figure 3.2. Reliability plot $R(t) = e^{-\lambda t}$.

Tok programu

- Procesorové systémy - procesor, pamäť, program, okolie
- Tok programu (*Control flow*) - postupnosť inštrukcií vo vykonávanom programe
- Graf toku programu - reprezentácia toku programu pomocou grafu
- Základný blok - časť programu bez vetvení a iných inštrukcií riadenia (okrem poslednej inštrukcie bloku)

Graf toku programu (2)

$$P = \{V, E\}$$

$$V = \{v_1, v_2, \ldots, v_n\}$$

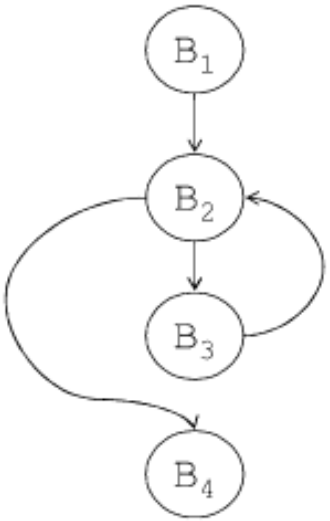
$$E = \{e_1, e_2, \ldots, e_m\}$$

v_i - základný blok

e_i - vetvenie $br_{i,j}$ z v_i do v_j

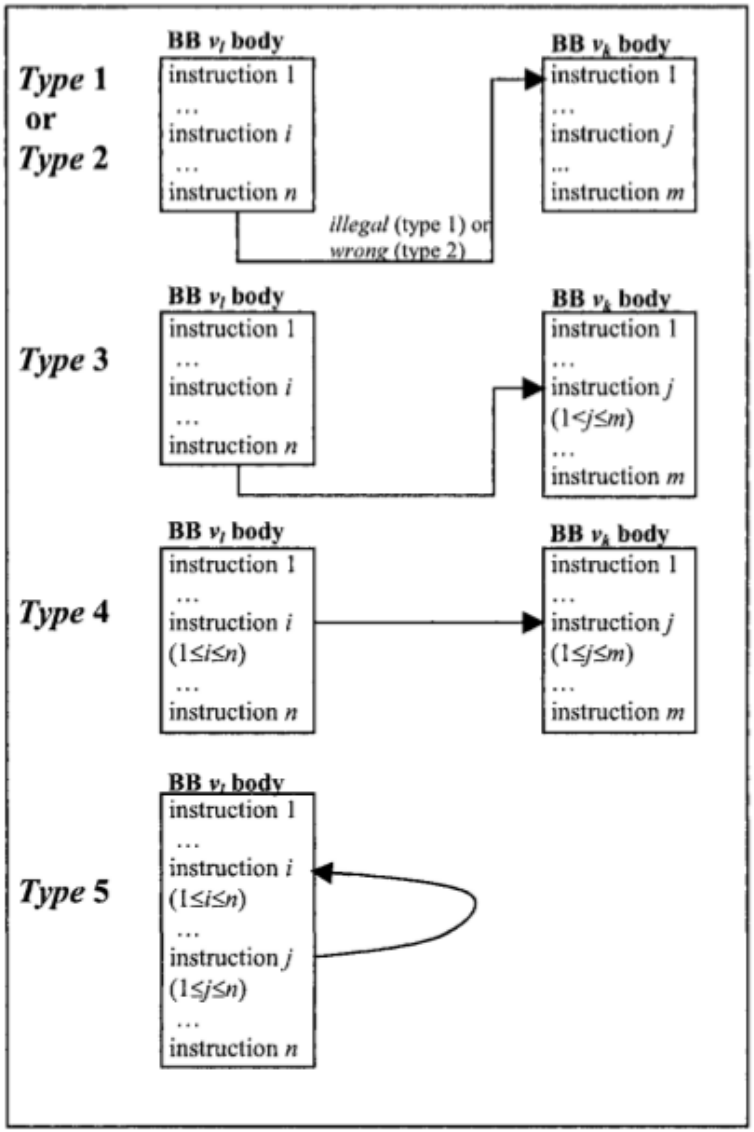
Ak $br_{i,j} \notin E$, vetvenie/skok je nelegálny

```
B1: x = 1;
    y = 5;
    i = 0;
B2: while( i < 5 ) {
B3:    z = x+i*y;
       i = i+1;
    }
B4: i = 2*z;
```



Chyby v toku programu

- typ 1 - nesprávny skok z konca bloku na začiatok iného bloku
- typ 2 - legálny ale nesprávny skok z konca bloku na začiatok iného bloku
- typ 3 - skok z konca bloku do stredu nesprávneho bloku
- typ 4 - skok z ktorejkoľvek časti bloku do inej časti iného bloku
- typ 5 - skok z ktorejkoľvek časti bloku do inej časti



Chyby v toku programu (2)

- porucha vloženia inštrukcie vetvenia (*branch insertion error*) - ak sa v základnom bloku zmení bežná inštrukcia na inštrukciu vetvenia a porucha vedie k vetveniu
- modifikácia cieľa vetvenia (*branch target modification error*) - cieľ inštrukcie vetvenia je zmenený na inú adresu z dôvodu poruchy a vetvenie sa vykoná
- vyradenie vetvenia (*branch deletion error*) - porucha spôsobí, že inštrukcia riadenia toku programu (vetvenie, skok, . . .) sa zmení na inštrukciu bez riadenia toku

Pri hard RT systémoch - časový alebo logický odklon od pôvodného toku programu

Metódy kontroly toku programu

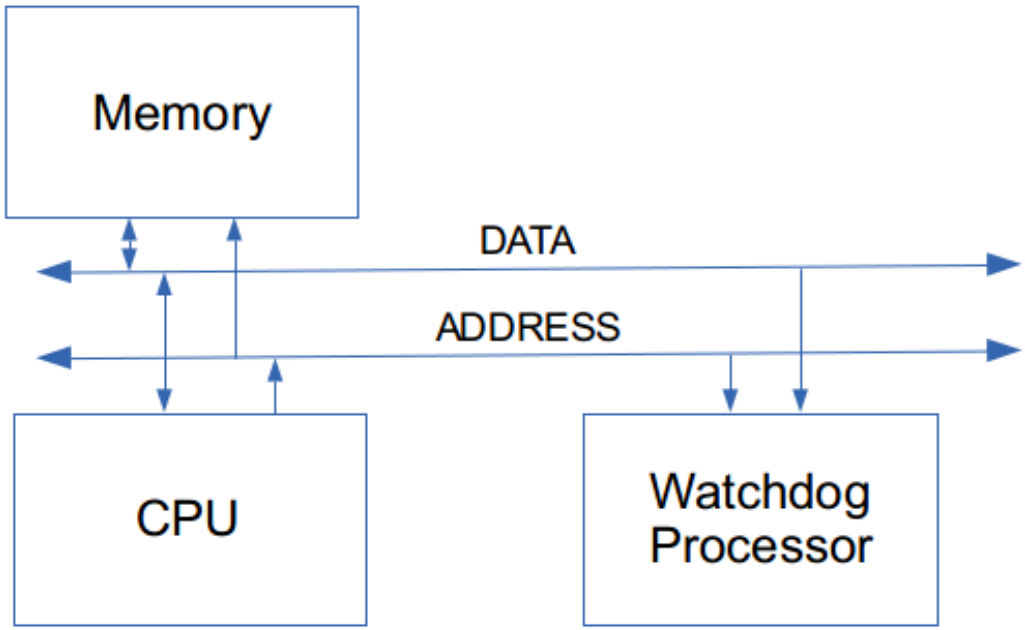
1. rozdelenie:
   - Hardvérové metódy
   - Softvérové metódy
     - pre jazyky vyššej úrovne C, C++, ...
     - pre jazyky na úrovni assembleru
   - Hybridné metódy

2. rozdelenie:
   - metódy aplikovateľné na systémy s COTS procesormi
   - metódy využívajúce IP jadrá

Pôvodná hardvérová metóda

Watchdog procesor (r. 1988)

- rozšírenie pôvodnej myšlienky *watchdog* časovača



Watchdog procesor (WDP)

- kontrola toku programu
- monitorovanie zbernice medzi CPU a pamäťou
- overovanie, či je spustený správny blok programu a ich poradie
- WDP musí mať referenčné informácie o programe z grafu toku programu
- programové bloky majú pridelené príznaky
- detekcia porúch:
  - chybná inštrukcia (napr. zle načítaná z pamäte)
  - zlý skok v programe

WDP (2)

WDP:

- aktívny - počíta príznaky, vykonáva tieňový program
- pasívny - sleduje informácie na zbernici a kontroluje správnosť s referenčnou pamäťou

Dva typy prístupov KTP s WDP:

- statické prideľovanie príznakov
- odvodenie príznakov z inštrukcií
  - aj kontrola integrity
  - problém pri spracovaní inštrukcií

Problémy:

- moderné CPU s Cache pamäťou
- pamäťový nárast
- zvýšené oneskorenie
- problém pri superskalárnych strojoch
- modifikácia vykonávaného programu - nárast v pamäti

WDP (3)

Program + príznaky

```
.
.
send(50);
read(input, number);
send(187);
Begin
repeat
      send(-82);
      if number < 0 then begin


         send(-12);
         negsum := negsum + number;
         send(28);
         number := number * -1;
      End else send(-13);
      send(155);
      Sum := sum + number;
      send(48);
read(input, number);
until (number = 0) or (eoln(input));
send(-83);
end;
.
.
```

Program WDP

```
.
.
if signature <> 50 then error;

if signature <> 187 then error;
   Begin

if signature <> -82 then error;
   repeat
(* loop terminated when signature
different than -82 *)
      if signature = -12 then begin
      (* -12 means 'if' executed *)
        if signature <> 28 then error;

   end;
   if signature <> 155 then error;

   if signature <> 48 then error;

until signature <> -82;

end;
.
.
```

Figure 5-5. Labeled Structured Program (left) and Structural Reference Program (right).

Softvérové metódy

SIHFT - softvérovo implementovaná hardvérová odolnosť

- určené pre systémy s COTS procesormi
- používa sa príznaková analýza
- model programu pomocou grafu toku programu (*control-flow graph*)

CFCSS

s_i - príznak pridelený každému bloku
G - *run-time* príznak meniaci sa počas vykonávania
príznaková funkcia f:

$$f(G, d_i) = G \oplus d_i$$

kde

$$d_i = s_j \oplus s_i$$

a s_j - príznak predchodcu v_i.

Hybridné a nové hardvérové metódy

Každá z nových metód využíva softvérovú podporu aspoň čiastočne.
Prednosti v nových hybridných a HW metódach:

- využitie systémov na čipe
- miniaturizácia
- používanie IP jadier
- návrh špecializovaného hardvéru

Príklad: CFCET - *Control-flow checking using execution tracing* (využitie ladiaceho portu)

Metóda CFCET (2006)

- Pentium procesory majú jednotku *Execution tracing unit*
- Generovanie špeciálnych zbernicových cyklov
- Extrahovanie grafu skokov (GTP) $\Rightarrow$ externý WDP s asociatívnou pamäťou
- Modul OS, ktorý zapne generovanie špeciálnych cyklov na zbernici
- Predpoklad použitia aj pri *Multi-tasking OS*

Výhody:

- Aplikovateľná metóda na COTS procesory s prúdovým spracovaním a *on-chip cache*
- Nie je potrebná modifikácia programu
- Detekcia chýb takmer bez oneskorenia

STU FIIT

CFCET (2006) (2)

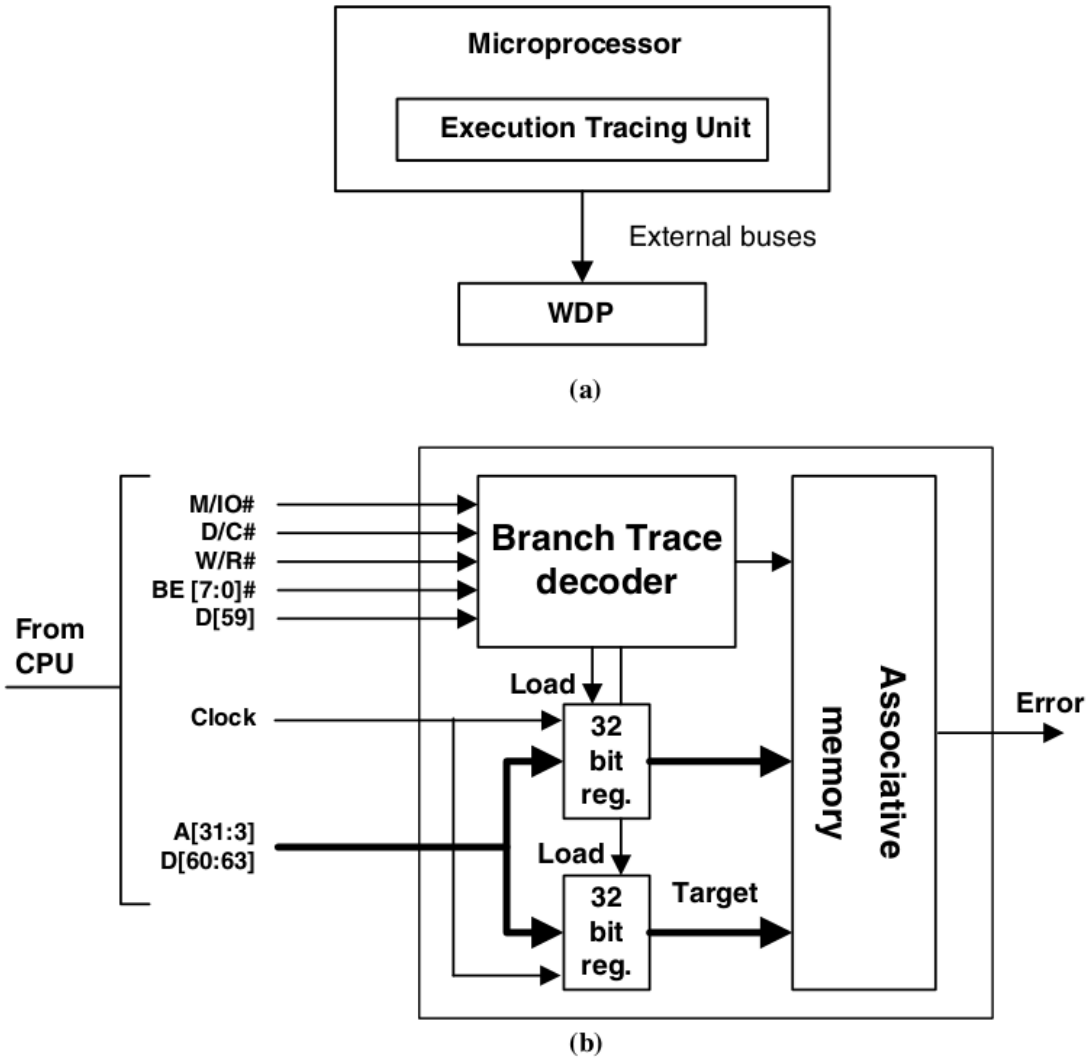


Fig. 3. (a) CFCET organization and (b) CFCET WDP organization.

Atribúty metód KTP

Hlavnými atribútmi metód kontroly toku programu:

- výkon systému
- pamäťový priestor
- plocha čipu pri HW a hybridných metódach
- pokrytie porúch
- oneskorenie detekcie porúch

Literatúra

[1] Goloubeva, O., Rebaudengo, M., et al.: Software-Implemented Hardware Fault Tolerance. Springer 2006, ISBN 978-0-387-26060-0.

[2] Koren, I., Krishna, C., M.: Fault-Tolerant Systems. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007, ISBN 978-0120885251.

[3] Oh, N.; Shirvani, P.P.; McCluskey, E.J.: Control-flow checking by software signatures, IEEE Transactions on Reliability, vol.51, no.1, pp.111,122, Mar 2002.

[4] Rajabzadeh, A., Miremadi, S.-G.: A hardware approach to concurrent error detection capability enhancement in COTS processors. In Proceedings of the 11th Pacific Rim International Symposium on Dependable Computing, pp. 8, Dec. 2005.

[5] Dubrova, E.: Fault-Tolerant Design: An Introduction, KTH Royal Institute of Technology, Stockholm (Sweden),2008. URL:http://web.it.kth.se/~dubrova/draft.pdf.

STU FIIT