Opora „Dependabilita softwarových a hardwarových systémů“

Doc. Viktor Maškov DrSc.

OBSAH

Úvodní slovo

V kurzu budou uvažovány otázky dependability jak softwarových, stejně tak i hardwarových systémů. Budou probrány základy vývoje spolehlivého softwaru. V kurzu se uvažují metody hodnocení spolehlivosti softwarových a hardwarových systémů. Studenti se seznáni s metodami a prostředky pro zlepšení spolehlivosti softwarových a hardwarových systémů a pro zajištění jejich odolnosti proti závadám. Uvažují se taky otázky modelování spolehlivosti softwarových a hardwarových systémů.

Kapitola 1. Úvod do problematiky

Cíl kapitoly:

- Vysvětlit vznik a význam pojmu a koncepce „Dependability".
- Upřesnit místo a role hardwaru v poskytování dependabilní služby.

Klíčová slova : Dependabilita, atributy dependability, spolehlivost, dostupnost hardwaru.

Výkladová část :
Vznik pojmu a koncepce „Dependability" složitých systémů se vysvětluje. Uvádí se místo a role hardwaru v poskytování dependabilní služby.
Jedna z prvních definic dependability má následující tvar:
„Dependabilita je schopnost výpočetního systému poskytovat službu, na níž se lze spolehnout".
Postupem času byla koncepce dependability prostudována (především z pohledu praktického využití a kvantitativního hodnocení) a objevily se i další alternativní definice jako například:
„*Dependabilita systému je schopnost systému vyhnout se takovým selháním, jejichž četnost výskytu, resp. závažnost, by byla větší než úroveň přípustná pro uživatele*".
Formálně je dependabilita interpretována jako jedná z vlastností systému a má své samostatné specifikace a standardy. Na rozdíl od specifikací věnovaných funkčnosti a výkonnosti systému, stanovuje specifikace dependability požadavky ke každému jednotlivému atributu dependability.
V současné době existuje „Technický výbor 56 : Dependability" v IEC (tj. International Electrotechnical Commission, www.iec.ch), který navrhuje a udržuje mezinárodní standardy v oboru dependability.
Standardy navržené IEC TC 56 poskytují metody a nástroje pro hodnocení dependability a pro údržbu zařízení, služeb a systémů v průběhu jejich života.
Atributy dependability odrážejí jak kvantitativní tak kvalitativní hodnocení dpendability systému.
V současnosti se uvádějí tyto primární atributy dependability:

- dostupnost: pohotovost k provedení korektní služby;
- spolehlivost: kontinuita poskytování korektní služby;
- zabezpečení: absence katastrofických následků pro uživatele a prostředí;
- důvěrnost: absence neautorizovaného prozrazení (odhalení) informací;
- integrita: absence nevhodných změn stavu systému;
- udržovatelnost: schopnost podstoupit opravy a modifikace.

Je nutno poznamenat, že pouze „dostupnost" a „spolehlivost" mohou být kvantitativně vyhodnoceny. Ostatní atributy jsou pouze kvalitativní a do jisté míry závisí na subjektivním posuzování.
Kromě primárních atributů dependability existuje i celá řada atributů sekundárních, které jsou buď odvozeny z atributů primárních (a jsou tudíž vyjádřitelné pomocí primárních atributů) nebo jsou zaměřeny na určité hrozby pro dependabilitu (tj. hodnotí dependabilitu systému vzhledem k těmto specifickým hrozbám).
Mezi sekundární atributy prvého druhu patří například:

- zodpovědnost (dostupnost a integrita totožnosti osoby, která provádí určitou operaci v rámci služby);
- originalita (integrita obsahu zprávy včetně metadat, například času odeslání zprávy);
- nepopíratelnost (dostupnost a integrita totožnosti odesílatele nebo příjemce zprávy).

Typickým sekundárním atributem druhého druhu je robustnost. V širokém smyslu robustnost hodnotí dependabilitu vzhledem k externím závadám, tj. charakterizuje reakci systému na specifickou třídu závad. V užším pojetí je robustnost definována jako odolnost systému proti chybným vstupním datům.
Jednotlivé atributy dependability si lze představit jako fasety omezující hodnocení dependability konkrétního systému – nebo duálně jako fasety, které umožňují vyjádřit požadavek na dependabilitu libovolného systému.

V jednotlivých reálných situacích je kladen důraz na různé atributy, tj. fasety dependability. Preference jednotlivých faset ovlivňuje výběr technik, které bz měly být použity pro zajištění dependability, resp. k odvrácení hrozeb v dané oblasti.
Atributy, u nichž se nepředpokládají hrozby, nebo na něž není kladen dostatečný důraz, mohou být opomenuty (tj. nejsou uvažovány v návrhu systému).
Hardware jako součást složitého systému může do značné míry ovlivnit jeho dependabilitu. Tento vliv závisí na spolehlivosti a dostupnosti hardwaru.

Kontrolní otázky:
1. Definujte pojem „Dependabilita“.
2. Pojmenujte primární a sekundární atributy Dependability.
3. Charakterizujete výpočetní systém poskytující dependabilní službu.

Úkoly pro samostatnou práci:
Prostudovat sekundární atributy dependability. Použit literaturu: V. Mashkov, J. Fiser. Samokontrola a samodiagnostika na systémové úrovni. Lviv, Ukrainian Academic Press, 2010, 176 stran.

Kapitola 2. Spolehlivost softwarových systémů

Cíl kapitoly:
- Definovat pojem spolehlivost SW.
- probrat problém měření spolehlivosti SW.

Klíčová slova : spolehlivost, měření spolehlivosti SW, velikost SW, metriky.

Výkladová část :
Spolehlivost je často definována jako pravděpodobnost, že systém, vozidlo, stroj, zařízení, atd. bude vykonávat svou zamýšlenou funkci v daných operačních podmínkách po stanovenou dobu.
Modely spolehlivosti softwaru se používají k odhadu spolehlivosti nebo počtu zbývajících závad softwarového produktu, který byl uvolněn mezi zákazníky.
Důvody:
- objektivní vyjádření kvality produktu;
- plánování zdrojů pro fázi údržby softwaru.

Sledovanou proměnnou studovaných kriterií je počet závad (nebo rychlost nalezení závad normalizovaná počtem řádku kódu) za daný časový interval (týdny, měsíce, atd.), nebo doba mezi dvěma selháními.
Měření spolehlivosti SW zůstává složitým problémem, protože nemáme dobré pochopení povahy SW. Neexistuje jasná definice, k čemu se vztahuje spolehlivost SW. Je velmi těžké najit vyhovující postup měření spolehlivosti SW. Dokonce i taky zřejmé metriky SW, jako např. velikost SW nemá jednotné definice. V případě kdy není možné měřit spolehlivost SW přímo, je snaha měřit něco, co se přímo vztahuje na spolehlivost. Současná praxe měření spolehlivosti SW může být rozdělena na tři kategorii:
- metriky produktu (tj. SW a projektů);
- metriky projektového managementu;
- metriky procesu;
- metriky závad a selhání.

Velikost SW je chápana jako něco, co odráží složitost SW, návrhové úsilí a spolehlivost SW. Řádky kódu nebo tisíce řádků kódu (KLOC) je intuitivní postup pro měření velikosti SW. Ale neexistuje standardní postup pro spočítání. Typicky používáme zdrojový kód, ve kterém komentáře a neproveditelné příkazy nebudou započítané. Tato metoda nemůže přesně porovnávat složitost SW napsaného v různých programovacích jazycích.
Metrika funkcionálního bodu je metoda měření funkcionality míněného návrhu SW, která je založená na spočítání vstupů, výstupů, master souborů, dotazy a rozhraní. Master soubory obsahují popisná data, jako je jméno a adresa, stejně jako souhrnné informace, jako jsou prodeje splatná částka a od počátku roku k datu. Metoda může být použita pro hodnocení velikosti SW, kdy tyto funkce mohou být identifikovány. Je to měřítko funkcionální složitosti programu. Metoda dovoluje měřit funkcionalitu poskytovanou uživatele a je nezávislá na programovacím jazyce. Metoda se používá pro podnikové systémy. Není jasno, jestliže je použitelná pro vědecké aplikace a aplikace reálního času.
Složitost se přímo vztahuje k spolehlivosti SW, a proto je důležité prezentovat složitost SW. Metriky, které jsou zaměřeny na složitost, umožňují zjištění složitosti struktury programu pomocí zjednodušení kódu do grafické reprezentace. Příkladem reprezentativní metriky je McCabe’s Complexity Metric.

Metriky pokrytí testem jsou způsob jak ohodnotit závadu a spolehlivost pomocí testování SW produktů. Metriky jsou založené na předpokladu, že spolehlivost SW je funkce části softwaru, který již byl úspěšně verifikován.
Cílem metriky sběru závad a selhání je zjištění podmínek kdy SW bude správně fungovat. Minimálně budou shromažďovány údaje o počtu odhalených v průběhu testování závad (tj. před dodáním zákazníku) a o selháních (nebo jiných problémech) hlášených uživateli. Dále tyto údaje budou analyzovány za účelem spočítání intenzity selhání, střední doby mezi selháními (Mean Time Between Failures), střední dobu do následujícího selhání (Mean Time To Failure). Dostupnost (pravděpodobnost, že v daném čase program pracuje správně) pak může být spočítána jako podíl MTTF/MTBF.

Kontrolní otázky:

- definujte pojem „spolehlivost softwaru“
- charakterizujte proces selhání softwaru
- popište problém měření spolehlivosti SW

Úkoly pro samostatnou práci:
Prostudovat různé metriky, které se používají pro měření spolehlivosti SW. Použit literaturu:
Mashkov V., Fiser J. Samokontrola a samodiagnostika na systémové úrovni. Lviv, Ukrainian Acedemic Press, 2010, 176 stran.
Řepa V. Analýza a návrh informačních systémů. Praha, Ekopress, 1999.

Kapitola 3. Modely spolehlivosti SW

Cíl kapitoly:

- charakterizovat modely spolehlivosti SW.
- vysvětlit specifiku použití modelů spolehlivosti SW.

Klíčová slova : model spolehlivosti, předpověď spolehlivosti, intenzita závad.

Výkladová část :
Statický model spolehlivosti SW používá atributy projektu nebo programových modulů k odhadu počtu závad v softwaru. Parametry modelů jsou odhadovány na základě řady předchozích projektů.
Dynamický model používá průběžného vývoje vzorů závad k odhadu spolehlivosti finálního produktu. Parametry dynamických modelů jsou odhadovány na základě mnoha údajů zaznamenaných o hodnoceném produktu k danému datu.
Mnoho modelů spolehlivosti SW vzniklo poté, co lidé se snažili pochopit tomu, jak a proč SW může selhat. Je taky snaha kvantitativní ohodnotit spolehlivost SW. Více než 200 modelů spolehlivosti SW bylo navrženo od roku 1970 ale problém jak kvantifikovat spolehlivost SW je pořad převážně nevyřešený. Je nutné poznamenat, že jak existující modely, tak i modely, které právě teď vzniká, ještě nejsou schopny započítat složitost SW, omezení a předpoklady ohledně procesu vykonávající kvantifikování spolehlivost SW. Proto, neexistuje jednotný model, který by byl použitelný v kterékoli situaci. Žádný model je kompletní a reprezentativní. Jeden model může pracovat velmi dobře pro určitou skupinu SW, ale bude málo užitečný pro jinou skupinu SW.
Většina modelů SW obsahuje následující části: předpoklady, činitele a matematické funkce, které se vztahují ke spolehlivosti. Matematické funkce jsou obvykle exponenciální nebo logaritmické. Techniky modelování softwaru mohou být tříděny do dvou skupin: predikční modelování a hodnoticí modelování. Obě skupiny technik jsou založeny na pozorování a shromáždění dat o selhání a na analýzu pomocí statistik. Hlavní rozdíl mezi tyto modely (co se tyká jejich vztahu k spolehlivosti) je uveden v následující tabulce.

aspekt	Predikční modely	Hodnoticí modely
Časový rámec	Předpověď spolehlivosti někdy v budoucnu	Hodnotí současnou spolehlivost nebo spolehlivost v nejbližší době
data	Používá dřívější data	Používá nejnovější data získané při návrhu SW
V případě kdy se používá ve fázi návrhu	Obvykle před návrhem nebo testováním SW	Obvykle ve pozdní fázi poté, co budou obdrženy některé data

Predikční modely zahrnují: Musa's Execution Time Model, Putnam's Model and Rome Lab models TR-92-51 a TR-92-15 a další. Pomocí predikčních modelů je možné predikovat spolehlivost SW ve fázi jeho návrhu a v případě nutnosti je možné spolehlivost zvýšit.
Hodnoticí modely zahrnují: modely s exponenciálním rozdělením, model s Weibullovým zozdělením, Thompson a Chelson's modely.
První dvě skupiny modelů patří ke klasickým modelům (mají název – klasické modely hodnoticí počet závad/intenzitu závad). Druhé skupiny modelů (Thompson a Chelson) patří k modelům, které používají Bayesian intenzitu závad.
Vzhledem k velké složitosti SW každý model musí mít extra předpoklady. Pouze omezeny činitele mají být uvažovány. Většina modelů spolehlivosti SW ignorují proces návrhu softwaru a se zaměřují na výsledky – pozorujeme závady a/nebo selhání. Tím se snižuje složitost a dosahuje se abstrakce. Však, modely jsou používané pro určité situace a pro vyřešení určité třidy problémů.

Kontrolní otázky:
1. pojmenujte existující modely spolehlivosti SW.
2. charakterizujte predikční a hodnoticí modely.
3. vysvětlete, co znamená intenzita závad

Úkoly pro samostatnou práci:
Prostudovat modely spolehlivosti SW. Použit literaturu: Řepa V. Analýza a návrh informačních systémů. Praha, Ekopress, 1999.

Kapitola 4. Diagnostika závad

Cíl kapitoly:
- vysvětlit metody, které se používají pro diagnostiku závad SW.
- probrat různé koeficienty podobnosti a jejich použití v diagnostických programech.

Klíčová slova : diagnostika, bílá skříňka, černá skříňka, koeficienty podobnosti.

Výkladová část :
Jestliže verifikace ukáže, že systém nesplňuje stanovené vlastnosti, musí být provedeny další dva kroky: diagnostika závad, jež vedli k tomu, že vlastnosti systému se liší od stanovených, a následně odstranění (korekce) těchto závad.
Diagnostika softwarových závad, která se provádí v průběhu verifikaci výpočetního systému, se liší od diagnostiky, kterou používají v době nasazení systému. Důvodem k tomu je to, že vývojové prostředí umožňuje použití složitých automatických diagnostických technik pro lokalizaci softwarových závad. Automatická diagnostika na rozdíl od ručního ladění nepotřebuje značných úsilí pro lokalizace závad a se provádí za výrazně kratší čas.
Metody, které se používají pro diagnostiku (lokalizaci) softwarových závad, mohou být tříděné v závislosti na tom, jakou informace o softwarovém systému máme k dispozici (např. o jeho interní komponentní struktuře, o chování systému atd.). Hlavními třídami jsou:
- bílá skříňka (white box);
- černá skříňka (black box).

Příklady metod, které se patří k černé skříňce, jsou:
- lokalizace závad na základě spektru;
- nejbližší soused (angl. nearest neighbor);
- dynamické krájení (segmentování) programu (angl. dynamic program slicing);
- deltové ladění (angl. delta debugging).

Diagnostika na základě černé skříňky nepotřebuje znalosti interní struktury programu, interní logiky, struktury dat a interního chování a stavů systému. Stačí pouze zdrojový kód programu, tj. řádky kódu.
Uvážíme (probereme) metodu černé skříňky na příkladě lokalizaci závad na základě spektru. V daném případě budeme sledovat provedení příkazů programu.

Program

```
příkaz 1;
příkaz 2;
if (podmínka)
  {
```

```
    blok příkazů;
  }
příkaz 3;
  .  .  .  .
příkaz N;
```

Pod příkazem rozumíme volání funkce, početní operací, přiřazení hodnoty atd.
Při každém spuštění programu posoudíme, jestliže výsledek je správný či nikoli. Spuštění programu provedeme na různých vstupních datech. Dále je nutno pro každý příkaz nebo blok zaznamenat, jestliže tento konkrétní příkaz byl proveden, kdy výsledek byl nesprávný. Jinými slovy každý příkaz (blok) bude označen indikátorem (angl. flag). Obecné indikátor může být přidělen i každému řádku programu. Intuitivně můžeme očekávat, že příkaz, který má nejvíce indikátorů, bude mít také i největší pravděpodobnost toho, že je vadným (tj. tento příkaz způsobil nesprávný výsledek programu). V reálných diagnostických nástrojích (např. program spektrum [5]) rozhodování o vadném příkazu nebo bloku programu se provádí na základě koeficientu podobnosti (angl. similarity coefficient), který v integrované podobě (číslem) odráží pravděpodobnost události, že příkaz, blok příkazů nebo řádek programu obsahuje chybu.
Podstatu koeficientu podobnosti vysvětlíme na zjednodušeném příkladu s třemi bloky i třemi spuštěními programu. Předpokládáme, že při druhém spuštění výsledek programu se liší od očekávaného správného výsledku. Předpokládáme také, že máme k dispozici prostředky (např. speciální diagnostický program) pro registraci informaci o aktivitě komponent programu (resp. bloků programu). V daném případě blok 1 nebyl proveden při třetím spuštění a blok 2 nebyl proveden při prvním a třetím spuštění. Blok 3 byl proveden při všech spuštěních programu. Tuto informace o chování systému můžeme zobrazit v následující formě (viz Obr.X)

	příkaz1	příkaz2	příkaz3	výsledek
1. spuštění	1	0	1	0
2. spuštění	1	1	1	1
3.spuštění	0	0	1	0

Obrázek 8.1.. Aktivita bloků a výsledky programu pro tři spuštění

Koeficient podobnosti odráží podobnost sloupce, který koresponduje určitému bloku programu, sloupce, který koresponduje výsledkům spuštění programu. Tato podobnost binárních sloupců může být stanovena na základě

příkaz3		výsledek
1	⟺	0
1	⟺	1
1	⟺	0

porovnání jejich jednotlivých elementů. Porovnání provádíme pouze pro elementy, které se nachází ve stejném řádku sloupců

Označíme porovnání i ⇔ j jako p_{ij} , kde i,j∈{0,1}. Dále spočítáme, kolik různých druhů porovnání bylo provedeno, tj. spočítáme následující sumy:

$$S_{ij} = \sum p_{ij} \ , \ i,j \in \{0,1\}$$

Existují několik návrhů, jak na základě hodnot s_{ij} spočítat koeficient podobnosti. Každému jednotlivému návrhu odpovídá vlastní název koeficientu podobnosti (např. koeficient Jaccarda K_j, koeficient Tarantula K_t, koeficient Ochiai K_o, koeficient AMPLE K_a). Dané koeficienty spočítáme jako:

$$K_J = \frac{s_{11}}{s_{11} + s_{01} + s_{10}}$$

$$K_T = \frac{\frac{s_{11}}{s_{11}+s_{01}}}{\frac{s_{11}}{s_{11}+s_{01}}+\frac{s_{10}}{s_{10}+s_{00}}}$$

$$K_O = \frac{s_{11}}{\sqrt{(s_{11}+s_{01})(s_{11}+s_{10})}}$$

$$K_A = \left| \frac{s_{11}}{s_{01}+s_{11}} - \frac{s_{10}}{s_{00}+s_{10}} \right|$$

Výběr toho či jiného koeficientu podobností pro diagnostické účely závisí od několika faktorů a je celkem na rozhodnutí odborníka, který koeficient použít pro určitý SW systém. V uvažovaném případě koeficient Jaccarda pro blok 3 se rovna

$$K_J = \frac{1}{1+2} = \frac{1}{3} = 0.33$$

Stejným způsobem můžeme spočítat koeficient Jaccarda i pro dva zbývající sloupce, které korespondují blokům 1 a 2. Pro blok 1 koeficient Jaccarda je rovna 0.5 a pro blok 2 je rovna 1. Čím více koeficient Jaccarda tím více pravděpodobnost, že tento blok zapříčinil nesprávný výsledek (nebo problém) v programu. V uvažovaném případě blok 2 má největší koeficient Jaccarda, což znamená, že tento blok je nejvíce podezřelý v způsobení chybného výsledku programu.

Koeficienty podobností se taky používají i pro diagnostiku SW závad v distribuovaných aplikacích. V daném případě koeficient podobností odráží pravděpodobnost selhání komponentu systému nebo subsystému.

Realizace metody černé skříňky vyžaduje několik dodatečných zaopatřen. Zaprvé je nutno zajistit odhalení chyb, tj. ohodnotit výsledek každého spuštění programu. Zadruhé je nutno zaznamenat aktivitu každého příkazu (nebo bloku) programu v každém spuštění (tj. jestliže byl proveden). Zatřetí potřebujeme diagnostický program, který by spočítal koeficienty podobností a vyhodnotil také i důvěryhodnost výsledku diagnostiky (angl. Accuracy or duality of the diagnosis).

Odhalení chyb se liší pro různé druhy SW systémů (vestavený SW, SW aplikace běžící na různých platformách, distribuované SW systémy a webové aplikace).

Vestavený SW je obvykle vyvinout pro specifický HW, který se používá v autech, ve vlacích, na letadlech, v telefonech, robotech, televize, hračkách, systémech bezpečnosti, řízečcích zařízení atd. V případě vestaveného SW odhalení chyb se většinou provádí na základě kontroly předchozích a následujících podmínek a tvrzení-prohlášení (což znamená, že v určitém bodě programu platí určitá podmínka). Angl. assertion statements.

V případě SW aplikací běžících na určitých platformách odhalení chyb se zajišťuje pomocí mechanismu zachycení výjimek (např. dělení nulou, index pole, atd.).

V případě webových aplikací se používají také chybové hlášení (zprávy o vzniklé chybě) použitých protokolů pro přenos informaci (http, SOAP).

Kromě metody lokalizace závad na základě spektru, používají se i další metody černé skříňky jako nejbližší soused, dynamické krájení programu, delta ladění a další.

Metoda nejbližšího souseda nejdříve vybere spuštění, které produkovalo nesprávný výsledek a pak vypočítá spuštění (resp. vstupní data), při kterém budou provedené příkazy (bloky příkazů), které dohromady mají nejvíce podobné pokrytí kódu (angl. most similar code coverage) s tím pokrytím, které bylo při nepodařeném spuštění programu. Poté bude určeno množství příkazů, které byli provedené v nepodařeném spuštění, ale nebyli provedené v podařeném spuštění. Tito příkazy jsou považované za příčinu chyby.

Metoda dynamického krájení programu postupně zužuje oblast podezřelých příkazů (bloků) do množství příkazů, které ovlivňují výstup programu (např. výstupní proměnné).

Delta ladění provádí porovnání stavů systému po podařeném a nepodařeném spuštění. Při porovnání se zjišťuje příčina, která způsobila rozdíl ve stavech SW systému.

Diagnostické metody, které se patří k černé skříňce, nepotřebují detailní rozbor kódu programu, který může být velmi náročný, ale na druhou stranu tito metody nevždy umožňují přesnou lokalizaci závady. Obecné v případě černé skříňky přesnost lokalizaci závad závisí na mechanismu odhalení chyb a na počtu podařených i nepodařených spuštění programu.

Na rozdíl od metod černé skříňky, metody bílé skříňky vyžadují velké množství informace o interní komponentní struktuře systému a o jeho chování (tj. informace o implementaci systému). Diagnostika softwarových závad na základě metod bílé skříňky bere ohled na

- příkazy programu, jeho větví a na různé druhy cest v toku programu;
- interní logiku a na strukturu dat;
- interní chování a na stavy systému.

Rozlišují se následné metody bílé skříňky:

- tradiční metody, které používají model SW systému buď ve formě diagramu toku programu nebo model systému na základě BNF (angl. Bipartite Network Flow) a grafů syntaxe anebo model ve formě grafu přechodů stavů systému;
- objektové-orientované;
- komponentně-orientované.

Metody bílé skříňky využívá různé techniky pro lokalizaci SW závad jako například:

- Slicing technique (technika krájení). Tato technika identifikuje části (angl.constructs) zdrojového kódu, které mohou ovlivnit hodnotu proměnné v určitém bodu programu.
- Dicing technique (technika kostkování). Tato technika v porovnání s technikou krájení umožňuje zredukovat počet příkazů, které je nutno zkontrolovat pro lokalizaci závady.

Pro diagnostiku SW závad velmi důležitým je pojem „jednotka“. V tradičních metodách pod jednotkou rozumíme proceduru, funkce nebo dokonce i příkaz. Pro objektově orientovaný systém pod jednotku rozumíme třídu a pro komponentně orientovaný systém – komponentu. Komponenta je znovupoužitelný blok programu a může být kombinována s dalšími komponentami. Komponenty se mohou nacházet na jednom počítače a mohou být umístěná na různých místech (serverech) v případě distribuovaného systému i komunikovat jedena s druhou podle aplikační logiky. Příklady komponent jsou: tlačítko v GUI, hodinky SW aplikaci, rozhraní databáze atd. Komponenty běží uvnitř kontejneru. Příklady kontejnerů jsou web servery, aplikační servery, databázové servery atd.
V případě objektové orientované diagnostiky (bílá skříňka) znalost implementaci systému umožňuje provést potřebné volání privátných metod, zkontrolovat privátní vlastnosti tříd, ověřit interní volání.
Obvykle metody bílé skřínky se používá pro diagnostiku SW závad v relativně malých částí programu. Diagnostika celého systému je velmi náročná a vyžaduje hodně času. Kromě toho (mimo jiného), tento druh diagnostiky zahrnuje návrh testovacích případů, který do velké míry ovlivňuje důvěryhodnost a přesnost výsledků diagnostiky.
Po ukončení případné diagnostiky a odstranění závad ve fáze verifikace a validace SW systému, systém vstupuje do provozu (nasazení systému). Dobu provozu je možné rozdělit na dvě hlavní části a to na dobu údržby systému (kdy systém neposkytuje službu) a na dobu, ve které systém běží i provádí specifikované úkoly (kdy systém poskytuje službu). V době údržby systému diagnostika a odstranění závad vystupují jako korekční a preventivní údržba. Korekční údržba má za cíl odstranit závady, které způsobily jednu nebo více chyb, a byly odhaleny. Preventivní údržba má za cíl odhalit a odstranit závady ještě před tím než závady způsobí chyby v průběhu normálního fungování systému (běhu programu). Například, závady návrhu, které byly odhaleny v jiných podobných systémech.
V době, kdy systém běží i provádí specifikované úkoly (poskytuje službu), diagnostika a odstranění závad jsou součásti procesu, který zajišťuje odolnost systému proti závadám.
Protože ne všechny závady mohou být odstraněny ze systému, je zapotřebí provést předpověď závad.

Kontrolní otázky:

1. v čem spočívá diagnostika na základě černé skříňky.
2. charakterizujte koeficient podobnosti a jeho použití pro diagnostiku závad SW.
3. popište metody bílé skříňky a techniky, které se používá pro lokalizaci závad SW.

Úkoly pro samostatnou práci:
Prostudovat metody, které se používá pro diagnostiku závad SW. Použit literaturu: Mashkov V., Fiser J. Samokontrola a samodiagnostika na systémové úrovni. Lviv, Ukrainian Acedemic Press, 2010, 176 stran.

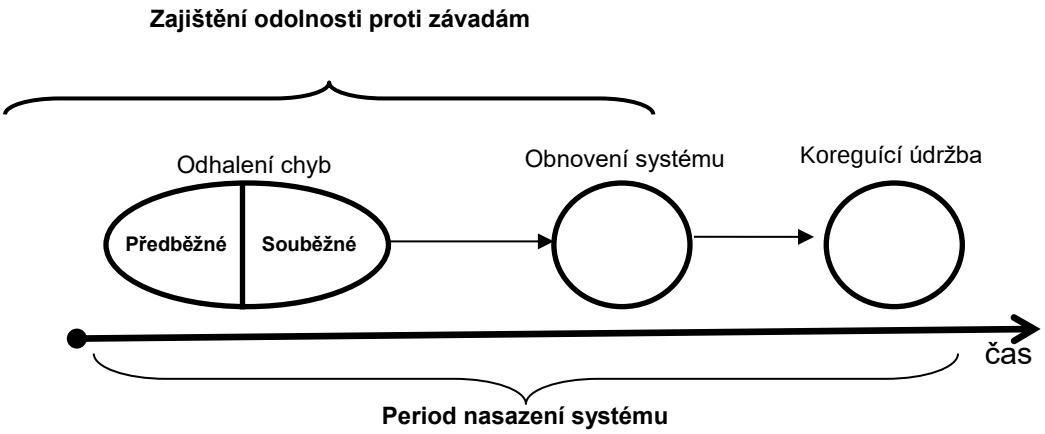
Kapitola 5. Odolné proti závadám SW architektury

Cíl kapitoly:

- vysvětlit hlavní kroky řešení problému zajištění odolnosti SW systémů proti závadám.
- podrobně rozebrat otázky odhalení chyb a následujícího obnovení systému.
- probrat hlavní schémata pro SW systémy

Klíčová slova : odolnost proti závadám, rozmanitost návrhu, schémata pro zajištění odolnosti SW proti závadám.

Výkladová část :
Mechanismy zajišťující odolnost proti závadám jsou zaměřeny na zabezpečení poskytování správné služby systémem za přítomnosti závad, které jsou v aktivním stavu. Základní časový průběh zajištění odolnosti systému proti závadám je ilustrován na obrázku



Obrázek. Odolnost systému proti závadám — časový průběh

Jak lze vidět, odolnost proti závadám je zajištěna dvěma dílčími procesy — nejdříve odhalením chyb a následným obnovením systému. O odhalení chyby v systému obvykle informuje buď signál nebo zpráva. Existují dvě třídy metod odhalení chyb: souběžné a předběžné. Souběžné odhalení chyb probíhá v průběhu poskytování služby systémem. Předběžné odhalení chyb probíhá v době, kdy je poskytování služby pozastaveno. V tomto případě je systém kontrolován na přítomnost latentních chyb a spících (neaktivních) závad.
Po úspěšném odhalení chyb musí následovat procedura obnovení systému. Obnovení transformuje systém ze stavu, který obsahuje jednu nebo více chyb a závad, do stavu bez odhalených chyb a závad, resp. bez závad, které by mohly být znovu aktivovány. Procesy odhalení chyb a obnovení systému se mohou v průběhu nasazení systému mnohonásobně opakovat.
Výše uvedené schéma presentuje zcela obecný pohled na proces zajištění odolnosti systému proti závadám. V konkrétním případě se mohou dílčí prvky procesu v čase překrývat, resp. nemusí být provedeny v celém svém rozsahu.
Konkrétní mechanismus *ošetření chyb,* tj. reakce na událost odhalení chyby, závisí v prvé řadě na principu implementace odolnosti systému proti závadám. V zásadě existují tři různé principy implementace tohoto procesu:
odrolování — transformace stavu probíhá jako vracení stavu systému do předem uloženého správného stavu (tj. např. stavu, který byl ještě před odhalením chyby).
přerolování — transformace stavu systému je přechodem do nového stavu, který neobsahuje chyby, které již byly odhaleny. Přerolování se nejčastěji spoléhá na nízkoúrovňový mechanismus výjimek a mechanismy z něho odvozené (např. ošetření vzniku souběžných výjimek).
maskování — transformace spočívá v odstranění chybného stavu systému tím, že chybné moduly jsou izolovány (tj. dále se neúčastní poskytování služby). To je možné pouze v systémech s redundancí prostředků (tj. s nadbytečnými prostředky, které by byly v případě bezchybných komponent zbytečné).
Součástí mechanismu ošetření chyb může být tzv. *ošetření závad,* tj. ošetření příčin chyb. Ošetření závad může obecně obsahovat následující kroky:
Krok 1: *Diagnostika závad*. Identifikace a zaznamenání příčiny chyb. V záznamech se uvádí lokalita a typ závady.
Krok 2: *Izolace závad*. Fyzické nebo logické vyloučení vadných komponentů z další účasti v poskytování služby (v případě logického vyloučení jde o vracení závad do spícího stavu).
Krok 3: *Rekonfigurace systému*. Buď přepnutí na náhradní komponenty, nebo použití správných komponent (které zůstaly v systému) a přerozdělení úkolů mezi správnými komponentami.
Krok 4: *Znovuinicializace*. Kontrola, aktualizace a zaznamenání nové konfigurace.
Po ošetření závad obvykle následuje korigující údržba, která odstraňuje závady, které byly izolovány v průběhu ošetření závad.
Využití mechanismu ošetření závad se však liší podle zvoleného principu implementace. V některých případech mohou být využívány jen některé kroky ošetření, resp. nemusí vůbec dojít k ošetření závad. Například v případě odrolování je běžně využita jen rekonfigurace systému (v podobě například tzv. obnovovacích bloků), ale v případě jednoduchého návratu k předem uloženému stavu, a tudíž i prostému opakování části programu, k ošetření závad vůbec nedochází (předpokládá se, že k opakovanému vzniku chyby nedojde).
Při využití mechanismu maskování lze využít diagnostiku závad pro lokalizaci závady, což umožňuje následné vyloučení závadné komponenty z poskytování služby (službu pak poskytují jen správné komponenty). K lokalizaci chybné komponenty však nemusí vůbec dojít, neboť postačující je zjištění správné komponenty, resp. podmnožiny správných komponent, které budou následně poskytovat službu. To znamená, že i v případě maskování nemusí být provedeno kompletní ošetření závad a závady mohou být důsledně ošetřeny a odstraněny až po delší době (např. u vestavěných systémů s dlouhodobým autonomním provozem). Dokončení diagnostiky se tak přesouvá až do fáze korigující údržby (kdy je nejčastěji zapotřebí účast externího agenta).

Při volbě konkrétní metody implementace odhalení a ošetření chyb, resp. ošetření závad je nutno vycházet z předpokladů o typu a charakteru selhání. Nejčastěji se používají následující (omezující) předpoklady:
systémy s ovladatelnými selháními:
systémy, jejichž návrh a implementace zajišťují, že tyto systémy selžou specifickým způsobem popsaným v požadavcích dependability a pouze do přijatelné míry. Příkladem ovladatelných selhání jsou:

- nesprávná ale stálá výchozí hodnota (opakem je nahodilá výchozí hodnota)
- absence výchozí hodnoty resp. signálu (ticho na rozdíl od „blábolení“)
- konzistentní selhání (opakem je nekonzistentní selhání).

systémy typu „selhání ⟶ zastavení“ :
selhání se vždy (resp. do přijatelné míry) jeví jako zastavení systému (např. projevující se jako „mlčení“ systému).
systémy s neškodnými selháními:
systémy, jejichž selhání jsou do přijatelné míry méně závažná.
Situaci dále komplikují problémy spojené s rekurzívností pojmu „odolnost proti závadám“. Je totiž důležité, aby i mechanismy, které zajišťují odolnost systému proti závadám, byly samy chráněny proti svým závadám, které samozřejmě mohou ovlivnit jejich činnost. Tento rekurzivní problém známý již od starověku v podobě otázky: „*Kdo stráží stráže*?“[I] musí být uvažován při výběru, resp. návrhu mechanismů zajišťujících odolnost systémů, neboť odolnost samotných mechanismů může do značné míry ovlivnit dependabilitu celého systému.
Důležitým rysem, resp. východiskem, většiny schémat obnovy v oblasti softwarových systémů je použití tzv. diverzity (rozmanitosti) návrhu. Princip diverzity návrhu je založen na používání několika (nadbytečných) variant návrhu a implementace (u softwaru kódu), čehož se využívá pro odhalení chyb a obnovení aplikace. Jednotlivé varianty jsou vytvořeny nezávisle (např. různými týmy), ale musí vyhovovat společné specifikaci služby. Jinak řečeno varianty musí poskytovat stejnou službu, ale musí být implementovány různými způsoby (což eliminuje nebo alespoň snižuje pravděpodobnost existence stejné závady). Adjudikátor (rozhodčí algoritmus) následně určí jeden výsledek (jenž je považován za správný) na základě výsledků různých variant.
Pro neparalelní softwarové systémy se používají následují tří hlavní schémata:

- obnovovací bloky (OB);
- n-variantní programování (NVP);
- n-samokontrolní programování (NVP).

Další schémata jsou (povětšinou) jen kombinacemi těchto schémat nebo jejich modifikacemi. Uvést lze např. distribuované obnovovací bloky, konsenzní obnovovací bloky, samokonfigurující optimální programování, certifikační stopy nebo t/(n-1) variantní programování.
U paralelních systémů (kooperačních i konkurenčních) se navíc používají mechanismy jako atomické akce, atomické transakce, rozšířené konverzace a koordinované atomické akce.
Přehled hlavních rysů těchto mechanismů najdete v následující tabulce.

Tabulka. Porovnání mechanismů odolnosti proti závadám

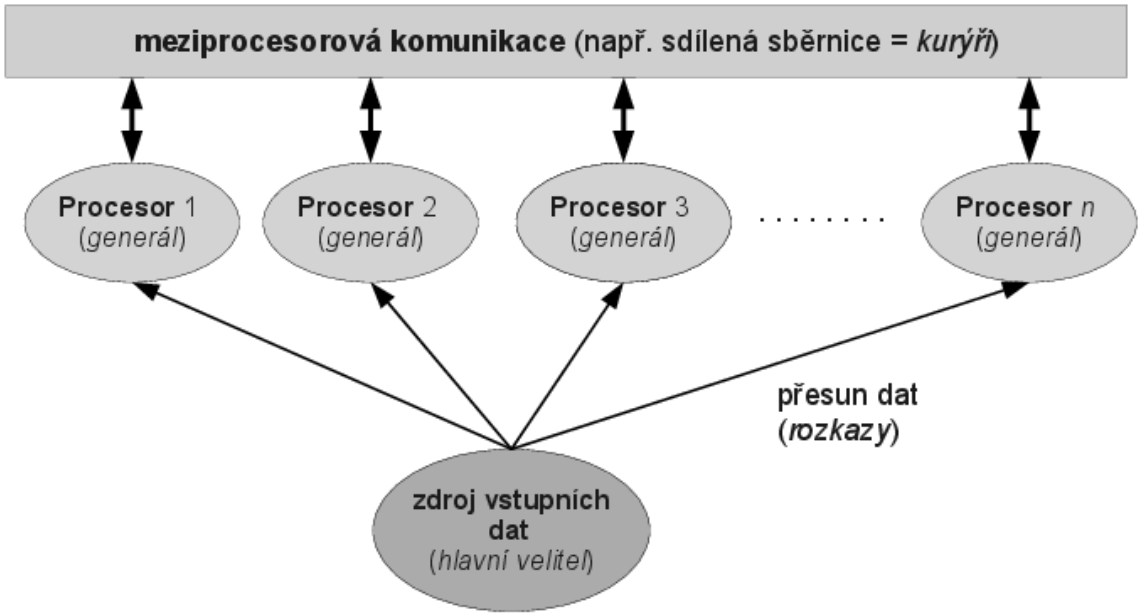
metoda	souběžnost	závady	obnovení
konverzace [1976]	kooperační	závady návrhu	odrolování (rozmanitost SW)
atomické akce [1986]	kooperační	závady prostředí, závady návrhu	přerolování (výjimky), odrolování (rozmanitost SW)
atomické transakce [1965]	konkurenční	závady hardwaru	odrolování (opakování)
rozšířené konverzace [1991]	konkurenční kooperační	závady návrhu	odrolování (rozmanitost SW)
koordinované atomické akce [1995]	konkurenční kooperační	závady prostředí závady návrhu závady hardwaru	přerolování (výjimky) odrolování (rozmanitost SW, opakování)

V rozsáhlých webových aplikacích je nutno uvažovat i nekonzistentní selhání způsobené napadením systému. U těchto systémů lze pro zajištění odolnosti využít *byzantských algoritmů*.
Byzantské algoritmy řeší tzv. byzantský problém, který lze obecně definovat za využití vojenské terminologie: Armáda řízená skupinou generálů obklíčí pevnost nepřátel. Každý generál je velitelem své vlastní vojenské jednotky a může se rozhodnout, jaké akce podnikne a jaké rozkazy vydá ostatním generálům (může tedy jednat nezávisle na ostatních generálech, může, ale nemusí uposlechnout jejich rozkazů). Generálové se domlouvají prostřednictvím kurýrů (poslů) a kromě rozkazů si mohou vyměňovat i další informace.

Generálové se musí dohodnout na společné (koordinované) akci. Klasickým příkladem takovéto akce je společný útok. Situace je však komplikována existencí zrádců, tj. generálů, jejichž cílem je narušení koordinace (tj. mohou provádět opačné akce a chybně informovat své partnery). Systém však navzdory aktivitě zrádců musí zajistit koordinaci mezi loajálními generály.
Přesněji řečeno všichni loajální (= nezrazující) generálové musí mít algoritmus, který by garantoval, že :

- všichni loajální generálové podniknou stejnou akci (zrádci mohou podniknout akci libovolnou)
- malý počet zrádců nesmí ohrozit plán, který provádějí loajální generálové.

Speciálním případem je situace, kdy rozkazy vydává pouze jediný generál (hlavní velitel). I ten však může být zrádcem. Pokud je však loajální, musí být akce vykonána loajálními generály v souhlasu s jeho rozkazem (jinak je vykonána akce opačná, resp. žádná). Je nutné si uvědomit, že loajalita hlavního velitele je definována jako konzistentní chování (velitel vydává stejné rozkazy a v rámci komunikace se chová konzistentně). Proradnost rozkazu (hlavní velitel může být ovlivněn nepřítelem), resp. jeho nezáměrnou chybnost, nelze v byzantském systému kontrolovat. Jinak řečeno loajální generálové musí vždy uposlechnout rozkaz konzistentně se chovajícího velitele.
Aplikace byzantského algoritmu ve výpočetních systémech může například spočívat v dosažení dohody o vstupních datech, která poskytuje vstupní komponenta (funkčně odpovídající hlavnímu veliteli), přičemž koordinovanou akcí je uložení (resp. nastavení) těchto dat u komponent-příjemců (odpovídají generálům). Komponentami mohou být např. procesory, servery replikovaných služeb, včetně např. replikovaných databází. Komponenty těchto systémů podléhají selháním, která jsou ve většině případů nekonzistentní (označuje se často jako tzv. byzantské selhání) a která odpovídají zrádcovskému chování ve vojenské terminologii.
Předpokládejme nejdříve systém s procesory, jež vzájemně komunikují a získávají data například ze vstupního portu, resp. IO procesoru (dále označeno jako zdroj dat), a produkují výstup, který by měl být u všech modulů identický (viz obrázek 9.2). Tento systém může obsahovat redundantní (nadbytečné) procesory, které by nebyly nutné při běžném zpracování dat v systému bez byzantských selhání.



Obrázek. Využití byzantských algoritmů na úrovni procesorů.

Toto schéma je obdobou běžnějšího schématu *většinového hlasování*, které zajišťuje, že je zvolen výstup, který převládá. Toto schéma však předpokládá, že získaná data jsou shodná, a tudíž správná (jinak řečeno zdroj dat je v kontextu byzantského problému loajální generál). Naproti tomu systém užívající byzantských algoritmů uvažuje i možnost selhání zdroje dat, a proto musí tudíž zahrnovat i vzájemnou komunikaci procesorů .
Podobně lze uvažovat i systémy s replikami serverů, kde vstupními daty jsou zprávy od primárního serveru. Zprávy v tomto systému mohou být pozměněny, resp. dokonce podvrženy útočníkem, mohou být duplikovány, resp. mohou docházet, se zpožděním a samozřejmě mohou být chybně interpretovány servery-replikanty. I přesto se servery musí dohodnout na společném výstupu, tj. na shodně poskytované službě.
Vlastní popis byzantských algoritmů je mimo rámec této knihy, je však možno uvést alespoň základní charakteristiku. Lze totiž dokázat, že pokud je n počet generálů (počet komponent) a f počet zrádců (selhavších komponent), pak řešení byzantského problému existuje jen v případě, pokud platí $n \geq 3f+1$.
Ve většině případů je však pro řešení problémů nekonzistence výhodnější použít samodiagnostiku, jíž jsme se zabývali v předchozích kapitolách. Samodiagnostika má potenciál odhalit větší počet různých chybových situací v systému a především rozlišit a identifikovat větší počet různých druhů selhání. Například v systému popsaném obrázkem 9.2 může samodiagnostika rozlišit selhání zdroje vstupních dat (tj. selhání IO procesoru) od selhání

komunikačního kanálu mezi procesory. Navíc může identifikovat, zda se jedná o selhání stálé, nahodilé nebo jen intermitentní. Za určitých podmínek může poskytovat důvěryhodný výsledek i pro systémy s větším počtem chybných komponentů (např. v případech, kdy je počet chybných komponentů větší než $\lfloor (n-1)/2 \rfloor$). Další výhodou je rychlejší reakce na změnu v architektuře systému např. v případě omezení komunikace nebo poklesu počtu komponent. Pružnější je i při využití nadbytečnosti (redundance), kterou může, ale nemusí využívat.

Kontrolní otázky:

1. jakým způsobem může být dosažena odolnost SW proti závadám.
2. jak mohou být odhaleny chyby v SW systému.
3. popište hlavní schémata pro zajištění odolnosti SW proti závadám.

Úkoly pro samostatnou práci:
Prostudovat schémata zajišťující odolnost SW proti závadám. Použit literaturu: Mashkov V., Fiser J. Samokontrola a samodiagnostika na systémové úrovni. Lviv, Ukrainian Acedemic Press, 2010, 176 stran.

Kapitola 6. Modely spolehlivosti hardwarových systémů

Cíl kapitoly:

- Popsat modely spolehlivosti HW.
- Vysvětlit použití různých modelů pro hodnocení spolehlivosti HW.

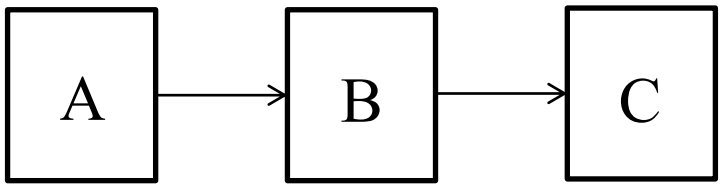
Klíčová slova : nadbytečnost, spojení komponent, režimy selhání.

Výkladová část :

Obvykle systém se skládá s velkého počtu komponent. Otázka zní jak spočítat spolehlivost systému, který obsahuje více než jeden komponent. V závislosti na tom jak komponenty systému jsou spojeny, můžeme definovat několik typů modelů spolehlivosti systému.
Mezi základní modely patří:

- sériové systémy
- aktivní nadbytečnost
- M-out-of-N nadbytečnost

Nejjednodušší model spolehlivosti je sériový model (viz Obr. 5.1), ve kterém všechny komponenty musí fungovat správně, aby celý systém fungoval správně.



Obrázek. Příklad sériového spojení komponent

Spolehlivost (v daném případě pravděpodobnost bezporuchového provozu) R_S je dána součinem pravděpodobností bezporuchového stavu všech jeho komponent

$$R_S = R_A * R_B * R_C$$

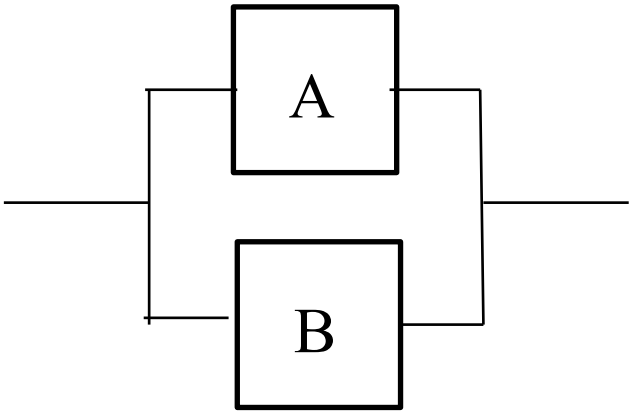
V případě exponenciálního rozdělení dob do poruchy všech komponent sériového modelu spolehlivost systému může být spočítána jako

$$R_S = e^{-\lambda_A t} \times e^{-\lambda_B t} \times e^{-\lambda_C t}$$

Intenzita poruch systému je dána

$$\lambda_S = \lambda_S + \lambda_B + \lambda_C$$

Jedním z nejběžnějších druhů nadbytečnosti je paralelní model spolehlivosti, ve kterém fungují dvě nezávislé jednotky systému. Celý systém bude fungovat správně, pokud jedna z jeho jednotek funguje správně. Na Obr. 5.2 je nadbytečný model pro dvě jednotky zobrazen.



Obrázek. Případ paralelního spojení komponent

Obvykle předpokládáme, že jednotlivé jednotky systému mají stejné spolehlivostní vlastnosti. V daném případě spolehlivost celého systému je daná

$$R_S = R_A + R_B - (R_A * R_B)$$

Za předpokladu, že intenzita poruch je konstantní spolehlivost systému může být spočítána jako

$$R_S = e^{-\lambda_A t} + e^{-\lambda_B t} - e^{-(\lambda_A + \lambda_B)t}$$

Tento výraz je platný pro neopravitelná zařízení.
U některých konfigurací paralelního modelu s nadbytečností potřebujeme minimálně *m* z *n* správných jednotek pro správné fungování celého systému. Spolehlivost takového systému označíme jako R_S a bezporuchovost jednotlivých komponent označíme jako R_X. Bezporuchovost celého systému odpovídá následujícímu vztahu

$$R_S = \sum_{i=m}^{n} C_i^N R_X^i (1 - R_X)^{N-i}$$

Je třeba poznamenat, že existují i další modely spolehlivosti systému, které jsou založené na použití nadbytečnosti.

Kontrolní otázky:
1. Charakterizujte základní vlastnosti sériového modelu.
2. Charakterizujte základní vlastnosti paralelního modelu.

Úkoly pro samostatnou práci:
Vyřešit základní příklady z oblasti spolehlivosti systémů (použit literaturu: S.R. Calabro. Základy spolehlivosti a jejich využití v praxi. Praha: SNTL, 1965).

Kapitola 7. Návrh spolehlivého HW

Cíl kapitoly:

- Popsat zásady pro návrh spolehlivého hardwaru.
- Popsat různé druhy nadbytečnosti.

Klíčová slova : Zvyšování spolehlivosti, využití nadbytečnosti.

Výkladová část :
Cílem návrhu spolehlivosti je navrhnout výrobek, který splňuje požadavky k jeho spolehlivosti v průběhu fungování v specifikovaném prostředí. K dosažení tohoto cíle je třeba dodržovat určitá pravidla. Existuje však několik obecných zásad, které by měly být použité. Zásady budou podrobně vysvětleny v průběhu výuky.
Obecně zajištění spolehlivosti hardwaru může být provedeno dvěma způsoby.
První způsob je pasivní zvyšování spolehlivosti, zatímco druhý způsob spočívá ve využití nadbytečnosti.
Mezi pasivní způsoby patří:

1) Použití vyzkoušených a dobře známých komponent;
2) Použití rozmanitosti aby vyloučil společné režimy selhání;
3) Zabezpečení vhodné volby pracovního režimu komponent;
4) Minimum složitosti;

5) Vhodná volba struktur systémů.

Předpokládá se, že bude použit minimální počet jednotek, který zajišťuje funkčnost systému.
Druhá možnost jak zajistit spolehlivost systému uvažuje různé typy nadbytečnosti.

Kontrolní otázky:
1. Pojmenujte několik obecných zásad pro návrh spolehlivého hardwaru.
2. Popište různé druhy nadbytečnosti.

Úkoly pro samostatnou práci:
Prostudovat zásady pro návrh spolehlivého hardwaru. Použit literaturu: . S.R. Calabro. Základy spolehlivosti a jejich využití v praxi. Praha: SNTL, 1965.

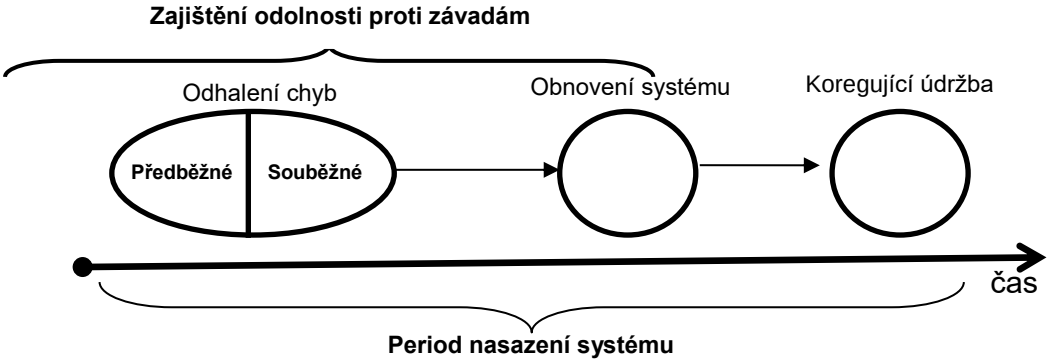
Kapitola 8. Odolnost proti závadám HW

Cíl kapitoly:
- Vysvětlit hlavní kroky řešení problému zajištění odolnosti proti závadám.
- Podrobně rozebrat otázky odhalení chyb a následujícího obnovení systému.

Klíčová slova : Odolnost proti závadám, odhalení chyb, obnovení systému, nadbytečnost, maskování, diagnostika.

Výkladová část :

Odolnost proti závadám je zaměřena na zabezpečení poskytování správné služby v přítomnosti závad, které jsou v aktivním stavu. Zajištění odolnosti systému proti závadám může být znázorněno pomocí následujícího obrázku



Obrázek. Proces zajištění odolnosti systému proti závadám.

Jak je to vidět, odolnost proti závadám se skládá ze dvou částí a to z odhalení chyb a následujícím obnovením systému. O odhalení chyby v systému obvykle informuje buď signál, nebo zpráva. Jsou dvě třídy metod odhalení chyb: souběžné a předběžné.
Souběžné odhalení chyb probíhá v průběhu poskytování služby systémem. Předběžné odhalení chyb probíhá v době kdy poskytování služby je pozastaveno. Tato metoda kontroluje systém na přítomnost latentních chyb a spících (neaktivních) závad.
Po úspěšném odhalení chyb následuje procedura obnovení systému. Obnovení transformuje stav systému, který obsahuje jednu nebo více chyb a závad, do stavu bez odhalených chyb a závad, které bychom mohli být znovu aktivovány. Procesy odhalení chyb a obnovení systému se mohou mnohokrát opakovat v průběhu nasazení systému.
Toto je obecné představení procesu, který zajišťuje odolnost systému proti závadám. Nemusí to znamenat, že všechny prvky tohoto procesu musí proběhnout v pořadí uvedeném na obrázku a v celém rozsahu. Existují tři různé principy implementaci procesu, který zajišťuje odolnost systému proti závadám a to na základě:
- odrolování, když transformace stavu probíhá jako vracení stavu systému do předem zachovaného stavu, který byl ještě před odhalením chyby.
- přerolování, když transformace stavu systému probíhá jako přechod do nového stavu, který neobsahuje již odhaleny chyby.
- maskování, když transformace spočívá v odstranění chybného stavu systému od účastí v poskytování služby.

Různé implementace do různé míry používá ošetření závad. Obecné ošetření závad skládá se ze 4 kroků:
Krok 1: Diagnostika závad. Identifikuje a zaznamenává příčiny chyb. V záznamech se uvádí lokalita a typ závady.

Krok 2: Izolace závad. Vykonává fyzické nebo logické vyloučení vadných komponentů z další účastí v poskytování služby (v případě logického vyloučení jde o vracení závad do spícího stavu).
Krok 3: Rekonfigurace systému. Buď přepíná na náhradní komponenty, nebo používá správné komponenty (které zůstali v systému) a přerozdělí úkoly mezi správnými komponenty.
Krok 4: Znovuinicializace. Kontrola, aktualizace a zaznamenání nové konfigurace.
Obvykle po ošetření závad následuje koregující údržba, která odstraňuje závady, které byli izolované v průběhu ošetření závad.
Reakce na událost odhalení chyby se projevuje (představuje, vystupuje jako, je) jako ošetření chyb. Tato reakce může zahrnout všechny nebo jenom některé kroky ošetření závad.
V případě použití odrolování, ošetření chyb může použit rekonfiguraci systému nebo může vůbec nepoužívat ošetření závad (jako v případě vrácení k předem zachovanému stavu a prostému (jednoduchému) opakovaní častí programu, která byla provedená posle zachování stavu (tj. nepovedených příkazů). V případě opakování ošetření závad může být provedeno po úspěšném ukončení ošetření chyb v době, kdy poskytování služby je pozastaveno.
Přerolování se obvykle spoléhá na mechanismus ošetření výjimek a může včlenit (začlenit) dodatečný mechanismus pro vyřešení četných výjimek, které vznikly souběžně v několika komponentech systému.
Maskování je zvláštně tím, že chybný stav systému resp. chybná komponenta systému je odstraněná od účastí v poskytování služby (viz Obr. 8.1) většinou bez provedení lokalizace chybné komponenty (stáčí pouze zjistit správný komponent resp. podmnožinu komponentů, které budou poskytovat službu). Což znamená, že maskování nevyžaduje okamžité ošetření závad. V případě vestavených SW systému s dlouhou dobou autonomního fungování (např. robot) závady mohou být ošetřené a odstraněné po další době. Vyloučení chybné komponenty systému z účastí v poskytování služby je možné kvůli použití nadbytečných (angl. redundancy) prostředků. Za podmínkou, že všechny komponenty systému jsou bezchybné, tyto prostředky byli by zbytečné.
Obvykle po ošetření závad následuje koregující údržba (viz Obr.X), která odstraňuje závady, které byli izolované v průběhu ošetření závad. Hlavní odlišující rysa koregující údržby spočívá v tom, že údržba potřebuje účast externího agenta.
Výběr metody odhalení chyb, ošetření chyb a ošetření závad, a jejich implementace se přímo vztahuje na základní předpoklad o selhání.
Systémy, jejichž návrh a implementace jsou takové, že tyto systémy selžou pouze specifickým způsobem popsaným v požadavcích k dependability a pouze do přijatelné míry, jsou systémy s ovladatelným selháním. Příklady ovladatelných selhání jsou:

- stále výchozí hodnota (nesprávná) na rozdíl od nahodilé výchozí hodnoty
- absence výchozí hodnoty (ticho) na rozdíl od „blábolení“
- konzistentní na rozdíl od nekonzistentního selhání.

Systém, jehož selhání se vždycky do přijatelné míry jeví jako zástava, se jmenuje systém s selháním typu: „selhání → zastavení“ nebo „selhání → mlčení“.
Systém, jehož selhání jsou do přijatelné míry méně závažná, se jmenuje systém s neškodnými selháními.
Odolnost proti závadám je rekurzivní pojem. Je důležité, aby mechanismy, které zajišťují odolnost proti závadám, byly sami ochráněny proti závadám, které mohou ovlivnit jejich činnost. Příklady jsou:

- replikace schématu hlasování
- samokontrolní kontroléry
- „stabilní“ paměť pro programy obnovení systému atd.

Táta rekurze vystupuje jako problém „straž nad stražím“ i musí být uvažovaná při výběru (návrhu) mechanismů, které zajišťují odolnost systému proti závadám, protože spolehlivost (resp. odolnost) těchto mechanismů může do značné míry ovlivnit dependability celého systému.
Existující schémata pro zajištění odolnosti systému proti závadám používají různé předpoklady o selhání systému (většinou jde o ovladatelných selháních) a do různé míry řeší problém „straž nad stražím“.

Kontrolní otázky:
1. Jakým způsobem může být zajištěna odolnost proti závadám složitých HW systémů.
2. Popište proceduru odhalení chyb.
3. Jak se provádí obnovení systému.

Úkoly pro samostatnou práci:
Prostudovat problém zajištění odolnosti proti závadám. Použit literaturu: . V. Mashkov, J. Fiser. Samokontrola a samodiagnostika na systémové úrovni. Lviv, Ukrainian Academic Press, 2010, 176 stran.

Kapitola 9. Odolnost proti závadám distribuovaných systémů

Cíl kapitoly:

- Vysvětlit podstatu odolnosti proti závadám distribuovaných systémů.
- Podrobně rozebrat metody umožňující zajistit odolnost proti závadám distribuovaných systémů.

Klíčová slova : Odolnost proti závadám, atomické transakce, replikace.

Výkladová část :
Jedním z prvních metod zajištění odolnosti proti závadám v distribuovaných systémech bylo aplikování objektového modelu a modelu atomické transakce.
Podle objektového modelu trvalý objekt se normálně nachází v pasivním stavu. Stav trvalého objektu je uložen v objektové paměti nebo objektové DB i může být aktivován požadavkem (tj. kdy bude vyvolán). Aktivace objektu znamená zavádění jeho stavu a metod z objektové paměti do nestabilní paměti a přidružení serveru pro přijímání RPC (dálkového volání procedury).
Atomické transakce jsou použité pro řízení změny stavů aktivovaných objektů. Vlastnosti atomických transakcí zajišťují průhlednost selhání.
Objekty mohou být přemístěny z jednoho uzlu do druhého, aby vylepšil odolnost proti závadám. Přemístění objektu není viditelné pro uživatele objektů. Jednoduchý ale velmi efektivní způsob jak zajistit migrace spočívá v tom, aby dovolit (umožnit) aktivaci objektu nejenom v uzlu, ve kterém objekt je uložen. Toto může být dosáhnuto když dovolím aby operace modulu podpory trvalých objektů byly vyvolány). ze vzdálených serverů objektů (tj. nejenom z lokálních serverů. Toto umožní serverům obdržet stav a metody objektu, který se nachází ve vzdálené objektové paměti. Takto uzel, který nemá objektové paměti, teď již může také spustit server objektu.
Pro zvýšení dostupnosti objektů mohou být použity několik replik objektu. Předpokládá se, že trvalý stav objektu se nachází v jedné objektové paměti uzlu. Takže, když uzel zhroutí, objekt bude nedostupný. Dostupnost objektu může být zvýšena pomocí replikace objektu na několika uzlech, což znamená, že stav objektu bude uložen ve více než jedné objektové paměti.
Takové objektové repliky musí být řízeny přes vhodný protokol zajišťující konzistentnost replik.
Existují tři aspekty řízení konzistence replik:

1) Vazba objektu;
2) Aktivace objektu a přístup k objektu;
3) Řízení replikami.

V průběhu výuky aspekty řízení konzistence replik budou podrobně rozebrány.

Kontrolní otázky:
1. Popište hlavní moduly distribuovaného systému.
2. Charakterizujte průhlednostní vlastnosti distribuovaného systému
3. V čem spočívá řízení replikami.

Úkoly pro samostatnou práci:
Prostudovat metody zajišťující odolnost proti závadám distribuovaných systémů. Použit literaturu: Parrington G.D., Shrivastava S.K., Wheater S.M., Little M.C. The design and implementation of Arjuna. (dostupná z: http://www.cs.newcastle.ac.uk/publications/articles/papers/95.pdf)

Literatura

základní:

1. Laprie, J.C., ed. *Dependability: Basic concepts and terminology- in English, French, German, Italian and Japanese*. Springer-Verlag, Vienna, Austria, 1992.
2. Mashkov V., Fišer J. *Samokontrola a samodiagnostika na systémové úrovni*. Lviv: Ukrainian Academic Press, 2010. ISBN 978-966-322-169-4.
3. Mashkov V. *Selected problems of system level self-diagnosis*. Lviv: Ukrainian Academic Press, 2011. ISBN 978-966-322-365-0.

Doporučená:

1. Aggarwal K. *Reliability Engineering*. Springer, 1993. ISBN 0-7923-2524-9.
2. Michael R. L. Handbook of Software Reliability Engineering. IEEE Computer Society Press and McGraw-Hill Book Company, 1996. ISBN 0070394008.