ZÁPADOČESKÁ
UNIVERZITA
V PLZNI

KIV/VSP - Průběžná práce

Diskrétní simulační modely

Příklad: 4 (zadání 3 / 7)

Jiří Kučera (A08N0092P)
Narozen 15. 2. 1985
kalwi@students.zcu.cz

Obecné zadání

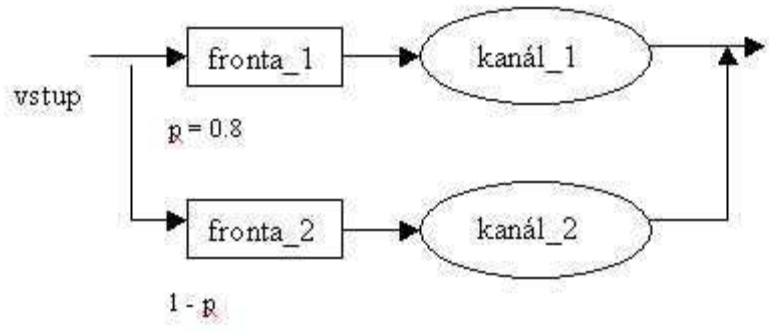
Ve všech řešených příkladech se provádí objektově orientovaná analýza modelu, která zahrnuje:

1. Určení typů objektů, které budou v modelu použity. Vystačíte se základními typy LINK (pasivní prvky seznamů), HEAD (seznamy) a PROCESS (samostatné výpočetní aktivity (vlákna) modelu) a od nich odvozenými vlastními typy. Doplňte slovní popis, k čemu budou zavedené typy sloužit.
2. Pro každý specifikovaný typ popis atributů a metod plus případný komentář. Pokud použijete přímo základní typ (třeba HEAD jde většinou použít bez modifikace), není popis atributů a metod třeba.
3. Data potřebná pro statistiku se snažte zapouzdřit jako atributy objektů. Teprve tehdy, když to jednoduše nepůjde, zaveďte je jako globální proměnné. Tip: různé statistiky jsou již obsažené v použitých simulačních knihovnách - a třeba z nich lze snadno dopočítat i další požadované statistiky
4. Pro každý typ odvozený od PROCESS uveďte též popis programu procesu (tj. co dělá) - slovně nebo lépe jako hrubý vývojový diagram.

Dále se provede programová realizace modelu s využitím programových nástrojů C-Sim, J-Sim nebo Cpp-Sim - viz. Studijní materiály.

Zadání

Pro zadanou otevřenou síť front určete L_q a T_q. Časové intervaly mezi vstupy požadavků mají exponenciální rozdělení s parametrem $\lambda = 4$, oba kanály obsluhy mají exponenciálně rozdělenou dobu obsluhy s parametry $\mu_1 = 4$, $\mu_2 = 80$.



Kolik procent z dlouhého časového intervalu sledování sítě bude fronta_2 prázdná?

Teoretický výpočet (stručně)

Zadané hodnoty:

λ = 4
$\mu_1 = 4$
$\mu_2 = 80$

Výpočet toků uzly:

$$\Lambda_0 = \lambda = 4$$
$$\Lambda_1 = p\Lambda_0 = 0{,}8 \cdot 4 = 3{,}2$$
$$\Lambda_2 = (1 - \mathrm{p})\Lambda_0 = 0{,}2 \cdot 4 = 0{,}8$$

Výpočet zatížení uzlů:

$$\rho_1 = \frac{\Lambda_1}{\mu_1} = \frac{3{,}2}{4} = 0{,}8$$

$$\rho_2 = \frac{\Lambda_2}{\mu_2} = \frac{0{,}8}{80} = 0{,}01$$

Výpočet středního počtu požadavků v systému:

$$L_{q_1} = \frac{\rho_1^2}{1 - \rho_1} + \rho_1 m_1 = \frac{0{,}8^2}{1 - 0{,}8} + 0{,}8 \cdot 1 = 4$$

$$L_{q_2} = \frac{\rho_2^2}{1 - \rho_2} + \rho_2 m_2 = \frac{0{,}01^2}{1 - 0{,}01} + 0{,}01 \cdot 1 = 0, \overline{01}$$

$$L_q = L_{q_1} + L_{q_2} = 4 + 0, \overline{01} = 4, \overline{01}$$

Výpočet doby odezvy:

$$T_q = \frac{L_q}{\lambda} = \frac{4, \overline{01}}{4} \doteq 1\ jednotka\ \check{c}asu$$

Výpočet pravděpodobnosti, že druhá fronta je prázdná:

$$p_0 = 1 - \rho$$

$$p_1 = p_0 \frac{\lambda}{\mu} = p_0 \rho = (1 - \rho)\rho$$

$$p = p_0 + p_1 = (1 - \rho) + (1 - \rho)\rho$$

$$p = (1 - 0{,}01) + (1 - 0{,}01) \cdot 0{,}01 = 0{,}9999 = 99{,}99\ \%$$

Objektová analýza a dokumentace

Aplikace je psána v Javě s použitím knihoven JSim.

Objektový návrh jsem se snažil koncipovat zaprvé tak, aby byla práce s minimálními úpravami použitelná jako základ pro velkou semestrální práci (tzn., aby návrh tříd byl co nejobecnější), a zadruhé, abych zapouzdřil původní třídy knihovny JSim takovým způsobem, aby více vyhovovaly konvencím objektového programování. Například konstrukci typu `prvek.vložit(fronta)` jsem převedl na `fronta.vložit(prvek)`.

Rozhraní `Junction`

- Třída, která implementuje toto rozhraní, může přijímat a zpracovávat požadavky. Obsahuje jedinou metodu:
  - `put()` – slouží k předání požadavku instanci dané třídy a jeho zpracování.
- Umožňuje snadné propojování prvků sítě nezávisle na jejich typu.

Třída `Transaction`

- Reprezentuje požadavek. Dědí od třídy `JSimLink` a momentálně neobsahuje žádnou funkcionalitu, je jen přípravou na velkou semestrální práci.

Třída `Generator`

- Dědí od třídy `JSimProcess`.
- Generuje požadavky a posílá je na výstup typu `Junction`. Mezi generováním požadavků čeká náhodně dlouhou dobu s exponenciálním rozdělením.

Třída `Server`

- Dědí od třídy `JSimProcess`.
- Z fronty vybírá požadavky a zpracovává je (odesílá na výstup typu `Junction`). Doba zpracování požadavků je simulována čekáním náhodnou dobu s exponenciálním rozdělením.
- Ke každé instanci je automaticky přiřazen objekt typu `Queue`, který slouží jako vstupní fronta.

Třída `Queue`

- Implementuje rozhraní `Junction` (metoda `put()` slouží k vložení prvku do fronty).
- Zapouzdřuje třídu `JSimHead` a obsahuje metody pro obecnou práci s frontou:
  - `isEmpty()` – slouží ke zjištění, zda-li je fronta prázdná.
  - `first()` – výběr požadavku z fronty.
- Tuto třídu nelze instanciovat samostatně, každý server si sám vytvoří vlastní instanci.

Třída `Fork`

- Implementuje rozhraní `Junction`.
- Slouží jako křižovatka s jedním vstupem a dvěma výstupy.
- Požadavky přicházející na vstup posílá se zadanou pravděpodobností na jeden z výstupů.

Třída `Terminator`

- Implementuje rozhraní `Junction`.
- Nedělá nic – slouží jako zakončení sítě, na které je možno napojit ostatní objekty posílající požadavky na výstup typu `Junction`.

Třída `Main`

- Spouštěcí třída.
- Obsahuje definici sítě (propojení prvků), spouští simulaci a vypisuje výsledky.

Podrobný popis tříd a metod je uveden v přiloženém Javadocu.

Výsledky simulace

10 000 požadavků

```
Lq: 3.8571483781410634
Tq: 0.9642870945352658
druha fronta prazdna (%): 98.89936469046829
```

100 000 požadavků

```
Lq: 4.035763274733517
Tq: 1.0089408186833793
druha fronta prazdna (%): 98.9738333005265
```

1 000 000 požadavků

```
Lq: 3.980355563678933
Tq: 0.9950888909197333
druha fronta prazdna (%): 98.99790092173934
```

Závěr

S navrženou strukturou objektů nebyl problém sestavit síť podle zadání. Výsledky simulace jsou si podobné pro jakýkoliv ze sledovaných počtů a odpovídají teoretickým hodnotám. V provedených simulacích se L_q od teoretických hodnot liší méně než přibližně o ± 0.2, T_q o ± 0.04 a procenta doby, po kterou byla fronta prázdná, o ± 1.