Pravděpodobnostní modely počítačů

Stanislav Racek

O b s a h

K a p i t o l a 1

Ú v o d

Skriptum je zejména určeno jako základní literatura pro předmět *Výkonnost a spolehlivost číslicových systémů* využívaný v oboru *Informatika a výpočetní technika* (IVT). V rámci předmětu jsou prezentovány metodiky vytváření *matematických a simulačních modelů* využitelných k odhadu (predikci) výkonnostních a spolehlivostních ukazatelů číslicových a počítačových systémů [1].

V první kapitole neformálně vymezíme vybrané základní pojmy a principy, které jsou využívány v dalším textu. S terminologií z oblasti *modelování systémů* se lze též seznámit v lit. [Kind80], [Dou81], [Dou90], [Ráb82], [Neu88].

Reálný objekt

Tímto pojmem rozumíme určitou část reálného světa, více či méně přesně ohraničenou. *Okolím* objektu rozumíme zbývající část reálného světa. Reálný objekt má nějaké *vlastnosti*, označované též jako *atributy*.

Model (v širším smyslu)

V nejobecnějším možném pohledu budeme pod pojmem *model* rozumět jakékoliv účelově konstruované napodobení reálného objektu, označovaného v této souvislosti jako *originál* [2]. Modely mohou mít nejrůznější formu (např.

[1] Ve skriptech je uváděna pouze malá množina příkladů potřebných pro pochopení výkladu. Další rozsáhlejší množina příkladů (včetně zdrojových programů) určená pro cvičení a samostatnou práci je k dispozici na sdíleném adresáři serveru dostupného studentům. Simulační nástroj C-Sim využívaný k realizaci diskrétních simulačních modelů je dostupný na www stránce `http://www.c-sim.cz`

[2] Objekt s charakterem originálu nemusí fyzicky existovat, modely můžeme konstruovat

jiný reálný objekt, matematický či jiný formální i neformální popis originálu, simulační program realizovaný v nějakém počítači ap.). V procesu konstrukce modelu zpravidla uplatňujeme:

zjednodušení - tj. bereme v úvahu jen ty vlastnosti originálu, které jsou významné vzhledem k účelu konstrukce modelu,

zobecnění - tj. nemodelujeme jeden konkrétní reálný objekt (tj. *instanci*) ale celou *třídu* podobných reálných objektů.

Zjednodušení a zobecnění reality bývá zvykem označovat jako *abstrakci*. Zvolená úroveň abstrakce musí odpovídat účelu konstrukce modelu [3].

Systém

V technických (i dalších) vědách ve většině případů koncipujeme model jako *systém. Pod pojmem systém rozumíme účelově definovaný soubor prvků, mezi nimiž existují určité vztahy* [4]. Definování systému nad reálným objektem vyžaduje (kromě dříve uvedeného *zjednodušení* a *zobecnění*) ještě *přesný popis* (např. matematický) uvažovaných vlastností prvků a vazeb prvků.

Model s charakterem systému dále zavádí přesné vymezení *rozhraní* mezi reálným objektem a jeho okolím. Vlastnosti (veličiny), kterými objekt ovlivňuje okolí jsou označovány jako *výstupní*. Vlastnosti, které jsou stanoveny vlivem okolí jsou označovány jako *vstupní* [5].

Uvedeme některá možná třídění systémů podle různých hledisek:

- **Dynamické** versus **statické** systémy podle toho, zda některé vlastnosti prvků mají charakter *paměti* (tj. zda závisí či nezávisí na minulé posloupnosti změn jiných vlastností - tzv. *historii*). U dynamických systémů se tedy předpokládá jejich existence (a změny vlastností) v čase.

i pro dosud neexistující (např. ve stadiu návrhu) či domnělé objekty.

[3] Příliš složitý model je nákladný a jeho použití těžkopádné, příliš jednoduchý model nepostihuje dostatečně přesně chování originálu.

[4] Existuje řada přesnějších definic, pro potřeby modelování však tato obvykle postačuje. Prvkem rozumíme dále nedělitelnou (z hlediska použité rozlišovací úrovně) část systému. Soubor prvků v systému (a jejich vazby) se může s časem měnit.

[5] Nad konkrétní třídou reálných objektů můžeme zavést systém různými způsoby. Přitom vstupní vlastnosti chápeme jako *příčiny* ovlivňující chování objektu, výstupní (ovlivněné) vlastnosti považujeme za *následky*.

Všechny vlastnosti s charakterem paměti popisují *stav* systému. Pro analýzu časového vývoje vlastností dynamického systému musíme znát jeho *počáteční stav*, tj. stav v časovém bodě s hodnotou 0 (počátek pozorování). Počáteční stav je úplným popisem historie systému. Časový vývoj vlastností systému též bývá označován jako *chování* systému. V rámci těchto skript se budeme zabývat pouze modely dynamických systémů.

- **Spojité** versus **diskrétní** dynamické systémy podle toho, zda se časově závislé vlastnosti mění ve spojitém čase nebo v diskrétní množině bodů časové osy. V rámci těchto skript se zabýváme výhradně diskrétními systémy. S diskrétními systémy je spojen pojem *událost*. Tímto pojmem míníme každou změnu některé z uvažovaných vlastností v konkrétním časovém bodě, dále začlenění či vypuštění určitého prvku do resp. ze systému. Určitá událost může být příčinou dalších událostí v budoucích časových bodech.

- **Uzavřené** versus **otevřené** systémy podle toho, zda je uvažována interakce systému s okolím. V rámci těchto skript jsou uvedeny příklady otevřených i uzavřených systémů.

- **Stochastické** versus **deterministické** systémy podle toho, zda mají či nemají některé vlastnosti charakter náhodných veličin nebo procesů. V těchto skriptech jsou využívány zejména stochastické systémy, dále uváděné metodiky diskrétní simulace ale lze využít i pro deterministické (např. logické sekvenční) systémy.

Modelování

Modelováním rozumíme celý proces konstrukce a využití modelu. Jedná se o činnost, která vede k získání informací o *systému–originálu* prostřednictvím *systému–modelu.*

Tento proces by měl zahrnovat i etapu *verifikace* modelu, kdy ověřujeme, zda výsledky získané řešením modelu odpovídají výchozímu popisu modelu (tj. například zpětně dosazujeme získané řešení do výchozí rovnice). Verifikací ověřujeme tzv. *správnost* (angl. *correctness*) modelu. Formální (exaktní) verifikace vyžaduje formální (zpravidla matematický) popis prvků modelu a jejich vazeb.

Validitou modelu rozumíme „míru přiblížení“ modelu k originálu (nakolik přesně vystihují informace získané z modelu chování originálu). *Validita* je tedy „silnější“ pojem než *správnost*, protože dostatečnou blízkost k originálu můžeme docílit jen správným modelem. K ověření validity potřebujeme porovnávat výsledky experimentů prováděných s modelem s daty získanými měřením vlastností reálného originálu.

Pomineme–li fyzikální modely (např. malé letadlo v aerodynamickém tunelu), lze modely rozdělit do dvou hlavních kategorií:

Matematické modely jsou založeny na exaktním matematickém popisu vlastností a vazeb prvků originálu (např. soustavou diferenciálních rovnic). Lze je dále dělit na modely *analytické*, s kterými lze pracovat symbolickými matematickými postupy a získat *řešení v uzavřeném tvaru* (tj. jako vzorec postihující závislost vybraných vlastností) a modely *numerické*, které k řešení využívají numerických technik. Vlastnosti originálu jsou modelovány prostřednictvím matematických veličin a funkcí. Matematické modely se obecně vyznačují vysokým stupněm zjednodušení a tedy větší „mezerou“ mezi chováním modelu a originálu. Na druhé straně lze zejména u analytických modelů exaktně ověřit správnost modelu (*formální verifikace*) a lze jednoduše analyzovat závislost výsledků na parametrech modelu.

Simulační modely. Simulačním modelem rozumíme program vykonávaný na počítači s *cílem napodobit chování originálu.* Simulační program vytvoříme buď *přepisem numerického matematického modelu do programovacího jazyka* nebo *přímým modelováním časových změn struktury a vlastností prvků systému v modelovém čase.*

Při vytváření simulačního modelu druhého uvedeného typu (dále *strukturní model*) se obvykle využije objektový přístup. Prvky (instance) kategorizujeme do *tříd* a společné vlastnosti popisujeme pouze jednou pro celou třídu. Vlastnosti prvků modelujeme prostřednictvím *strukturovaných dat*, *funkcí* (též *operací* či *služeb* poskytovaných prvkem pro jiné prvky) a *aktivit* prvků (probíhají v čase, jsou označovány též jako *procesy*). Vazby prvků typicky modelujeme prostřednictvím dat s charakterem *odkazů.*

Výhodou simulačních modelů je zejména možnost zahrnout libovolnou úroveň detailů vlastností originálu (pochopitelně za cenu složitějšího

a hůře analyzovatelného modelu) a zmenšit tak rozdíly mezi chováním originálu a modelu (tj. zlepšit validitu modelu). Naproti tomu nelze simulační model exaktně verifikovat a k získání závislostí výsledků na parametrech modelu je třeba (často časově náročné) opakování mnoha výpočtů simulačního programu s různými vstupními daty.

V rámci těchto skript je kladen důraz zejména na využití simulačních modelů. Matematické modely (například *markovské náhodné procesy* nebo *stochastické petriho sítě*) jsou popisovány jen uživatelsky s cílem poskytnout *základní prostředky pro verifikaci simulačních modelů.*

Simulace

Podle formulace O. – J.Dahla (jeden z norských autorů **SIMULY**) *je simulace výzkumná metoda, jejíž podstata spočívá v tom, že zkoumaný dynamický systém nahradíme jeho simulátorem a s ním provádíme pokusy s cílem získat informaci o původním zkoumaném systému* [Kind80]. Simulátorem je nejčastěji simulační program spolu s počítačem, na kterém je vykonáván. V takovémto pojetí je *simulace* užší pojem než *modelování*, jedná se o modelování vyžadující provádění experimentů k získání výsledků (na rozdíl od např. využití analytických matematických modelů). Proces simulace typicky zahrnuje následující etapy:

Formulování dostatečně přesného popisu originálu (přesnost popisu odpovídá účelu použití modelu). Při strukturním modelování se využívají metodiky objektově orientované analýzy (tj. hledají se typové prvky, jejich vlastnosti a vazby).

Programování popisu ve zvoleném programovacím jazyce. Zasvěcenou publikací týkající se charakteristik simulačních programovacích jazyků je [Kind80]. V rámci těchto skript je využíván programovací nástroj **C-Sim** (knihovna funkcí a maker) [Rac93], [Rou95], který rozšiřuje možnosti jazyka **C** ve směru idejí programovacího jazyka **SIMULA** [Dahl72], [Stau78].

Experimentování se simulačním programem (je některými autory chápáno jako *simulace* v užším smyslu). Zahrnuje etapu částečné verifikace simulačního modelu (typicky realizovanou množinou experimentů, jejichž výsledky lze určit také prostřednictvím (jednoduchého)

matematického modelu konstruovaného pro tentýž originál a vzájemně porovnat). Další etapou je používání verifikovaného modelu a zpracování užitečné informace získané použitím modelu. Máme–li k dispozici data z reálného originálu, můžeme po jejich porovnání s daty získanými z modelu posoudit validitu modelu.

Uvedený postup pochopitelně nemusí proběhnout zcela přímočaře, například při zjištění větších rozdílů mezi výsledky modelu a realitou můžeme změnit až výchozí popis originálu a celý proces opakovat.

Základní úrovně modelů využívaných pro modelování číslicových a počítačových systémů

Úrovně modelů využívaných k vyšetřování chování číslicových a počítačových systémů se liší úrovní abstrakce, tedy úrovní podrobností zahrnutých do modelu. Přitom modely s nízkou úrovní abstrakce se hodí pro vyšetřování chování malých (relativně) částí počítače, modely s nejvyšší úrovní abstrakce se hodí k popisu chování počítačového systému jako celku. Obvykle se rozlišují následující úrovně modelů:

Fyzikální úroveň. Modely této úrovně se využívají k vyšetřování fyzikálních polí (např. teplotní pole, elektromagnetické pole) a jiných fyzikálních jevů (např. difuze částic v polovodičích) v součástkách a základních konstrukčních modulech počítače. Jako matematický model na této úrovni slouží *parciální diferenciální rovnice*, které se zpravidla řeší numericky *metodou sítí* nebo *metodou konečných prvků*. Nezávisle proměnnými v těchto rovnicích jsou prostorové souřadnice a pro nestacionární pole navíc i čas. S modely fyzikální úrovně je možné se při studiu IVT setkat zejména v rámci předmětů *Fyzika, Teoretická elektrotechnika, Elektronické součástky* a *Konstrukce počítačů*.

Úroveň elektrických obvodů. Modely této úrovně se typicky využívají k vyšetřování dynamického chování elektronických součástek (například zákmity na výstupech hradel při změně úrovně signálu). Abstrahujeme od prostorového rozložení fyzikálních polí a bereme v úvahu jen jejich integrální parametry (*kapacitu, indukčnost*). Parciální diferenciální rovnice předchozí úrovně takto degenerují na (jednodušší) obyčejné

diferenciální rovnice s časem jako nezávisle proměnnou. Matematickým modelem v této úrovni jsou tedy soustavy obyčejných diferenciálních rovnic. V simulačních modelech se tyto soustavy řeší numericky. S modely obvodové úrovně je možné se při studiu IVT setkat zejména v rámci předmětů *Fyzika, Teoretická elektrotechnika, Teorie obvodů, Základy měření* a *Konstrukce počítačů.*

Úroveň hradel a klopných obvodů. V této úrovni se dopouštíme dalšího zjednodušení reality - neuvažujeme dynamické chování elektronických součástek [6]. Spojité elektrické signály diskretizujeme do dvou úrovní (*logická 0*, *logická 1*). Fyzikální spojitý čas redukujeme na posloupnost časových bodů, ve kterých dochází ke změnám hodnot signálů (*diskretizace* času). Uvažujeme tedy pouze stacionární úrovně signálů na vývodech součástek a jejich změny v diskrétních časových bodech.

Matematickým modelem na této úrovni je *konečný automat.* Simulační modely jsou typicky konstruovány jako strukturní a umožňují vyšetřovat chování systémů složených z velkých počtů logických prvků (například velké *hradlové pole*). Automatové modely se využívají zejména v rámci předmětů *Teoretická informatika, Číslicové elektronické systémy* a *Logické systémy.*

Úroveň logických bloků. V modelech této úrovně se jako prvky využívají *registry, paměti, sčítačky, dekodéry* ap. Rozlišují se *datové, adresní* a *řídicí* logické signály, přičemž signály prvních dvou typů se sdružují do skupin. Na této úrovni neexistuje univerzální matematický model, což je dáno nesnadným jednotným matematickým popisem prvků a jejich vazeb. Využívají se ale strukturní modely spočívající v popisu *struktury* prostřednictvím vhodného jazyka (jazyky označované jako RTL - *Register Transfer Language*, viz například **Abel**). Takovýto popis lze použít jednak k simulaci chování popsaného logického systému a dále i k návrhu realizace systému (např. návrh masky pro výrobu IO, generování diagnostických testů obvodu ap.). Modely úrovně logických bloků se využívají zejména v rámci předmětů *Číslicové elektronické systémy* a *Architektury číslicových systémů.*

[6] Popřípadě jej shrnujeme do podoby jedné hodnoty s rozměrem času - *zpoždění* změny výstupu proti změně logické hodnoty vstupu.

Systémová úroveň. Modelem na této úrovni je nejčastěji *dynamický diskrétní stochastický systém*. Jako jeho prvky slouží hlavní části počítače (například *procesor*, *spojovací linka* či *vnější paměť*, popřípadě jednotlivé počítače v distribuovaném systému). Typicky se uvažují pouze *makrostavy* prvků (například *procesor je porouchaný* nebo *vyrovnávací paměť je prázdná*) a abstrahuje se od semantiky probíhajících výpočtů. Stavy prvků se (skokově) mění vlivem *událostí* přicházejících *asynchronně* v *náhodných časových bodech*. Modely se využívají s cílem předpovědět *výkonnostní* nebo *spolehlivostní* charakteristiky (ukazatele) systému, například *střední dobu bezporuchového provozu*, *střední délku fronty připravených procesů* nebo *střední dobu vyřízení dotazu v databázi*. Jako matematické modely se zde využívají například *petriho sítě* (jen statická analýza mrtvých stavů – viz dále), *stochastické petriho sítě* a *markovské náhodné procesy* (postihují i dynamiku náhodných přechodů mezi stavy). Simulační modely v této úrovni využívají *experimentální pravděpodobnostní modelování* (tj. generování a zpracování náhodných čísel) a dále popsané programovací techniky *interpretace událostí* a *pseudoparalelních procesů*. Modely systémové úrovně jsou často používány k preciznějšímu popisu (a analýze) chování *operačních systémů*, *počítačových sítí*, *komunikačních protokolů*, *výpočetních systémů odolných proti poruchám* ap.

Tato skripta se zabývají výhradně modely systémové úrovně s důrazem na simulační modely.

Kapitola 2

Matematické modely diskrétních stochastických systémů

2.1 Markovské náhodné procesy

Nejprve vysvětlíme některé používané základní pojmy. Obejdeme se přitom bez exaktních definic a důkazů, které lze v případě potřeby nalézt ve speciální literatuře, např. [Mand85], [Havr86], [Triv82].

Náhodný proces je každá funkce $X(t)$, jejíž hodnota při každé hodnotě argumentu je náhodná veličina. Jako argument t budeme dále s ohledem na aplikace uvažovat výhradně *čas*. Definičním oborem D argumentu t je (spojitá) množina nezáporných reálných čísel. Realizací náhodného procesu $X(t)$ rozumíme funkci $x(t)$ získanou konkrétním pokusem. Náhodný proces $X(t)$ můžeme rovněž chápat jako množinu všech možných realizací $x(t)$.

Náhodná posloupnost. Definiční obor argumentu D obsahuje konečný nebo spočetný počet hodnot, tedy $D = \{t_k\}$, $k = 0, 1, 2, ...$ Náhodná funkce $X(t)$ přechází v náhodnou posloupnost $X(t_k)$. Zkráceně ji zapisujeme $X(t_k) = X_k$.

Markovský řetězec je speciální náhodná posloupnost. Pravděpodobnost, že člen X_k posloupnosti nabude určitou hodnotu, je ovlivněna pouze hodnotou předchozího členu posloupnosti X_{k-1}. Bez ztráty na obecnosti můžeme dále uvažovat diskrétní čas t_k jako posloupnost nezáporných celých čísel, tedy $t_k = k$. V aplikacích představují hodnoty členů X_k čísla stavů modelovaného diskrétního systému. Jejich definičním oborem je například množina přirozených čísel $E = \{1, 2, ..., n\}$. Řetězec lze popsat *maticí pravděpodobností přechodů* mezi stavy

$$\boldsymbol{P}(k) = \begin{bmatrix} p_{11}(k) & p_{12}(k) & \ldots & p_{1n}(k) \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ p_{n1}(k) & p_{n2}(k) & \ldots & p_{nn}(k) \end{bmatrix}, \sum_{j=1}^{n} p_{ij}(k) = 1,$$

kde $p_{ij}(k)$ je pravděpodobnost přechodu ze stavu i do stavu j v diskrétním čase k. Jinak řečeno - je-li hodnota $X_k = i$, pak pravděpodobnost, že $X_{k+1} = j$, je $p_{ij}(k)$. Součet pravděpodobností v řádku musí být 1.

Diskrétní markovský proces. Množina argumentu D je spojitá (spojitý čas t), množina funkčních hodnot E je diskrétní. Uvažujeme opět $E = \{1, 2, ..., i, ..., n\}$. Hodnota náhodné funkce $X(t)$ má tedy význam čísla stavu a n je počet možných stavů. Ke změně stavu může dojít v libovolném časovém okamžiku, tedy pro jakékoliv t. Chování procesu po přechodu do stavu i (doba setrvání ve stavu, příští stav) nijak nezáleží na minulosti (posloupnosti stavů, kterými proces prošel). Jedná se o důležitou vlastnost markovských náhodných procesů označovanou jako absence paměti procesu (*markovská vlastnost*, angl. *memoryless property*). Diskrétní markovský proces je vhodným matematickým modelem pro stochastické systémy s diskrétní množinou stavů a spojitým časem.

Uvažujeme, že v čase t_1 je náhodný proces ve stavu i. Pravděpodobnost přechodu do stavu j v časovém intervalu $< t_1, t_2 >$ záleží zřejmě v obecném případě na hodnotách t_1, t_2 a tedy prvky matice přechodů jsou funkce $p_{ij} = p_{ij}(t_1, t_2)$. U *homogenních* diskrétních markovských procesů jsou pravděpodobnosti přechodů pouze funkcí časového rozdílu $\Delta t = t_2 - t_1$, a tedy $p_{ij} = p_{ij}(\Delta t)$. Dále budeme pracovat výhradně s homogenními markovskými procesy.

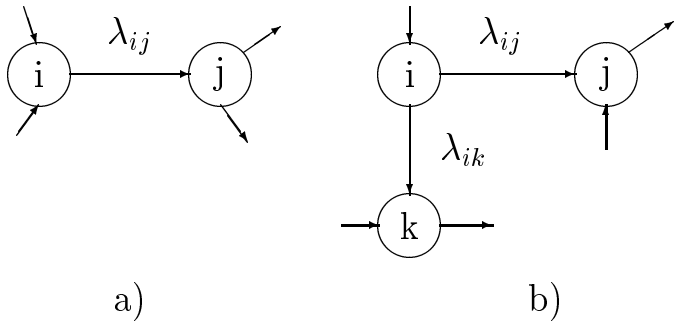
Důležitou veličinou u diskrétních markovských procesů je intenzita pravděpodobnosti přechodu ze stavu i do j, kterou označíme λ_{ij}. Intenzita pravděpodobnosti přechodu (zkráceně intenzita přechodu) je definována vztahem

$$\lambda_{ij} = \lim_{\Delta t \to 0} \frac{p_{ij}(\Delta t)}{\Delta t} \tag{2.1}$$

Pro homogenní markovské procesy je λ_{ij} konstantní. Pravděpodobnost přechodu ze stavu i do stavu j v dostatečně malém časovém intervalu Δt je pak

$$p_{ij}(\Delta t) \doteq \lambda_{ij}\ \Delta t \tag{2.2}$$

V aplikacích markovských náhodných procesů nás zpravidla zajímá, s jakou pravděpodobností se proces vyskytuje v čase t ve stavu i. Tuto pravděpodobnost označíme $p_i(t)$. Způsob jejího určení předvedeme na několika jednoduchých případech, které umožní pochopit obecný postup výpočtu. Budeme uvažovat konkrétní markovský náhodný proces s daným počtem stavů n. V čase $t = 0$ se tento proces vyskytuje ve stavu i a tedy $p_i(0) = 1$. Jediný stav různý od i, do kterého má přechod nenulovou intenzitu, je stav j. Uvažovaná situace je znázorněna na obrázku 2.1.



Obrázek 2.1: Fragmenty grafu přechodů

Podmíněná pravděpodobnost přechodu z i do j v malém časovém intervalu $< t, t + \mathrm{d}t >$ je podle (2.2) dána jako $\lambda_{ij}\mathrm{d}t$. Podmínkou je, že proces se v čase t nachází ve stavu i. Nepodmíněná pravděpodobnost přechodu je pak dána součinem podmíněné pravděpodobnosti přechodu a pravděpodobnosti podmínky, tj. $p_i(t)\ \lambda_{ij}\mathrm{d}t$. Přechodem uskutečněným v uvažovaném časovém intervalu $< t, t + \mathrm{d}t >$ se zmenšuje pravděpodobnost $p_i(t + \mathrm{d}t)$ proti $p_i(t)$ o $p_i(t)\ \lambda_{ij}\mathrm{d}t$. Tyto úvahy vedou ke vztahu

$$p_i(t + \mathrm{d}t) = p_i(t) - \lambda_{ij}\ p_i(t)\ \mathrm{d}t$$

Po úpravě postupně dostaneme

$$\frac{p_i(t + \mathrm{d}t) - p_i(t)}{\mathrm{d}t} = -\lambda_{ij} p_i(t)$$

$$p_i'(t) + \lambda_{ij}\ p_i(t) = 0$$

Získanou lineární diferenciální rovnici prvního řádu s konstantními koeficienty řešíme pro počáteční podmínku $p_i(0) = 1$ a známým postupem dostaneme výslednou pravděpodobnost setrvání ve stavu i.

$$p_i(t) = \mathrm{e}^{-\lambda_{ij} t}$$

Pravděpodobnost, že v čase t již proces nebude ve stavu i (tedy že náhodný čas odchodu ze stavu bude menší než t) je zřejmě distribuční funkcí $F_i(t)$ náhodné doby setrvání procesu ve stavu i

$$F_i(t) = 1 - \mathrm{e}^{-\lambda_{ij}t}$$

Jedná se o distribuční funkci exponenciálního rozdělení s parametrem λ_{ij}.

Zcela analogickým postupem dostaneme pro modifikovanou situaci znázorněnou na obrázku 2.1b diferenciální rovnici

$$p_i'(t) = -(\lambda_{ij} + \lambda_{ik})p_i(t) = -\lambda_i p_i(t)$$

Bez dalšího odvozování je možné učinit důležitý závěr, že *náhodná doba setrvání ve stavu* i *má exponenciální pravděpodobnostní rozdělení s parametrem* λ_i, *kde* λ_i *je součet intenzit všech odchodů ze stavu* i. Střední doba setrvání ve stavu (tj. střední hodnota exponenciálního rozdělení) i je pak $T_i = 1/\lambda_i$.

Na tomto místě zdůrazníme, že základním předpokladem využitelnosti markovského náhodného procesu jako modelu reálného systému je (kromě nezávislosti chování systému na jiném stavu než posledním) exponenciální pravděpodobnostní rozdělení doby setrvání v každém stavu systému (resp. exponenciální rozdělení časových intervalů mezi časovým bodem příchodu do stavu a vznikem událostí iniciujících jednotlivé přechody). Jinak řečeno: *nachází-li se model v čase* t *ve stavu* i, *je podmíněná pravděpodobnost realizace přechodu do jiného stavu* j *(tj. vzniku příslušné události) v navazujícím malém časovém intervalu* dt *stále stejná (tj. nezávislá na čase* t*) a daná vztahem* $\lambda_{ij}dt$.

Hrana vedoucí do stavu i například ze stavu h by zřejmě byla v diferenciální rovnici reprezentována prvkem $\lambda_{hi}\ p_h(t)$ s kladným znaménkem. Pro každý stav uvažovaného náhodného procesu (uzel grafu přechodů) můžeme tedy formulovat lineární diferenciální rovnici prvního řádu s konstantními koeficienty. *Matematickým popisem časového vývoje pravděpodobností stavů diskrétního markovského náhodného procesu je pak soustava lineárních diferenciálních rovnic v normálním tvaru, s konstantními koeficienty, doplněná vektorem počátečních pravděpodobností stavů.* Tato soustava se v teorii ná-

hodných procesů označuje jako druhý systém Kolmogorovových rovnic a lze ji maticově zapsat ve tvaru

$$\boldsymbol{p}'(t) = \boldsymbol{p}(t)\boldsymbol{\Lambda} \tag{2.3}$$

V uvedené maticové rovnici je $\boldsymbol{p}(t)$ řádkový vektor pravděpodobností stavů a $\boldsymbol{p}'(t)$ řádkový vektor derivací pravděpodobností stavů.

$$\boldsymbol{p}(t) = [p_1(t)\ p_2(t)\ ...\ p_n(t)] \tag{2.4}$$

$$\boldsymbol{p}'(t) = [p_1'(t)\ p_2'(t)\ ...\ p_n'(t)] \tag{2.5}$$

Čtvercová matice koeficientů soustavy (2.3) se nazývá matice intenzit pravděpodobností přechodů nebo zkráceně matice intenzit přechodů. Obecný prvek λ_{ij} matice $\boldsymbol{\Lambda}$ představuje pro $i \neq j$ intenzitu přechodu ze stavu i do stavu j. Řádkový součet prvků matice je nula a hodnota prvku v diagonále (pro $i = j$) je zápornou hodnotou součtu ostatních prvků v řádku, tedy $-\lambda_i$. Řešení soustavy (2.3) vyžaduje znalost vektoru počátečních pravděpodobností stavů $\boldsymbol{p}(0)$.

Pro další výklad je významný pojem *absorpční stav*. Pokud v grafu přechodů nevede ze stavu i žádná hrana do jiného stavu, je stav i absorpční. Pokud se proces do tohoto stavu dostane, nemůže již nabýt jinou hodnotu (končí v tomto stavu).

2.1.1 Markovské procesy s absorpčními stavy

Markovské náhodné procesy s absorpčními stavy se využívají zejména k modelování přechodných dějů. „Život“ modelu je časově omezený a končí dosažením absorpčního stavu. Typickou aplikací jsou spolehlivostní modely neobnovovaných systémů, tj. systémů, jejichž život končí neopravitelnou poruchou. Absorpční stavy modelu představují stavy, ve kterých je systém porouchaný jako celek. Použití markovských modelů ve spolehlivostních aplikacích je omezeno předpokladem, že intenzity poruch a oprav jsou konstantní pro všechny prvky systému. Náhodné doby do poruchy prvků a náhodné doby oprav prvků mají v tomto případě exponenciální pravděpodobnostní rozdělení.

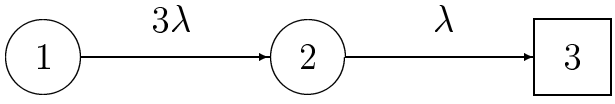
Postup konstrukce a využití markovského modelu s absorpčními stavy předvedeme na jednoduchém příkladu. Vzhledem k tomu, že existující programové systémy řeší numerickou stránku využití modelu i pro relativně

velký počet stavů (řádu tisíce, viz přehled v [Hlav92]), zaměříme se zejména na konstrukci modelu chápanou jako přechod od reálného problému ke grafu přechodů (nebo matici intenzit přechodů $\mathbf{\Lambda}$) odpovídajícího markovského procesu.

Příklad 2.1

Uvažujme číslicový systém složený ze tří shodných modulů s charakterem procesorů. Po uvedení do provozu pracují všechny tři procesory tak, že synchronně vyhodnocují stejnou vstupní posloupnost. Výstupy procesorů jsou sloučeny na výsledný výstup v majoritním modulu způsobem, který umožňuje tolerovat chybu na výstupu jednoho procesoru. Po trvalé poruše některého procesoru je jeden ze zbývajících dvou neporouchaných odpojen za účelem snížení celkové intenzity poruch. Zbývající procesor pokračuje v činnosti. Jeho porucha pak představuje poruchu celku. Intenzita poruch jednotlivých procesorů je konstantní a má hodnotu λ. Intenzita poruch se zpravidla udává v jednotkách hod^{-1} (např. $\lambda = 10^{-7}\ hod^{-1}$) a *číselně představuje pravděpodobnost, že se dosud neporouchaný modul porouchá v nejbližší hodině*. Intenzitu poruch majoritního modulu neuvažujeme (resp. ji pokládáme za zanedbatelnou).

Cílem analýzy modelu budiž určení pravděpodobnosti, že v obecném čase t bude modelovaný systém funkční. Tato veličina se v teorii spolehlivosti označuje jako *pravděpodobnost bezporuchového provozu*. Předpokládáme, že na počátku pozorování (tj. pro $t = 0$) určitě není porouchaný žádný modul systému. Graf přechodů použitého modelu je na obrázku 2.1. Ve stavu



Obrázek 2.2: Graf přechodů pro příklad 2.1

1 fungují všechny tři procesory. Do stavu 2 se dostaneme první poruchou procesoru a stav 3 představuje poruchu celku (absorpční stav). Intenzita přechodu $\lambda_{12} = 3\lambda$, protože se může porouchat kterýkoliv ze tří procesorů (tři možné přechody sdružíme do jednoho s trojnásobnou intenzitou). Intenzita přechodu $\lambda_{23} = \lambda$, protože jeden ze zbývajících neporouchaných

procesorů je po první poruše odpojen. Zapíšeme soustavu diferenciálních rovnic pro neznámé pravděpodobnosti stavů (viz odvozování rovnic v části 2.1).

$$\begin{aligned} p_1'(t) &= -3\lambda p_1(t) \\ p_2'(t) &= 3\lambda p_1(t) - \lambda p_2(t) \\ p_3'(t) &= \lambda p_2(t) \end{aligned}$$

Matice intenzit přechodů je pro tento případ

$$\mathbf{\Lambda} = \begin{bmatrix} -3\lambda & 3\lambda & 0 \\ 0 & -\lambda & \lambda \\ 0 & 0 & 0 \end{bmatrix}$$

Počáteční podmínky jsou $p_1(0) = 1$, $p_2(0) = 0$ a $p_3(0) = 0$, tedy všechny tři moduly v čase $t = 0$ fungují. Analytické řešení soustavy rovnic je jednoduché a s výhodou lze využít například Laplaceovy transformace. První rovnice je samostatně řešitelná s výsledkem

$$p_1(t) = \mathrm{e}^{-3\lambda t}$$

Po dosazení $p_1(t)$ do druhé rovnice můžeme opět samostatně řešit druhou rovnici s výsledkem

$$p_2(t) = \frac{3}{2}(\mathrm{e}^{-\lambda t} - \mathrm{e}^{-3\lambda t})$$

Hledaná pravděpodobnost bezporuchového provozu $R(t)$ je

$$R(t) = p_1(t) + p_2(t) = \frac{3}{2}\mathrm{e}^{-\lambda t} - \frac{1}{2}\mathrm{e}^{-3\lambda t}$$

2.1.2 Markovské procesy bez absorpčních stavů

Pokud nemá markovský proces absorpční stavy a graf přechodů je silně souvislý (tj. *existuje cesta z libovolného stavu do každého dalšího*), je *nerozložitelný* (přesnou matematickou definici tohoto pojmu lze nalézt např. v [Havr86]). Pro nerozložitelné markovské procesy je možné určit tzv. *limitní* hodnoty pravděpodobností stavů $p_1(\infty)$, $p_2(\infty)$, ... $p_n(\infty)$ jednodušším způsobem než na základě řešení soustavy diferenciálních rovnic (2.3).

Dále budeme kvůli přehlednosti označovat limitní pravděpodobnosti stavů jako $p_1, p_2, \ldots p_n$. Můžeme je určit na základě následující úvahy: *pokud existují limitní hodnoty pravděpodobností stavů, musí být odpovídající limitní hodnoty derivací nulové (funkční hodnota se již nemění).* Pro dostatečně velkou hodnotu času ($t \to \infty$) soustava (2.3) přejde na soustavu lineárních algebraických rovnic pro neznámé limitní pravděpodobnosti p_1 až p_n. V maticovém vyjádření tedy dostaneme rovnici

$$\mathbf{0} = \boldsymbol{p}\ \boldsymbol{\Lambda}, \tag{2.6}$$

kde $\boldsymbol{p} = \boldsymbol{p}(\infty) = [p_1\ p_2\ \ldots\ p_n]$ je hledaný vektor limitních pravděpodobností stavů a $\mathbf{0}$ je nulový vektor. V soustavě (2.6) je každá rovnice lineárně závislá na všech ostatních. Úspěšné vyřešení vyžaduje náhradu libovolné rovnice soustavy tzv. *normalizační podmínkou* $\sum_{i=1}^{n} p_i = 1$. Řešení opět vyjádříme maticovou rovnicí

$$\boldsymbol{p} = \boldsymbol{v}\ \boldsymbol{\Lambda}_m^{-1} \tag{2.7}$$

kde $\boldsymbol{\Lambda}_m$ je matice, která vznikne z $\boldsymbol{\Lambda}$ vyplněním libovolně vybraného i-tého sloupce samými jedničkami (doplnění normalizační podmínky). Vektor $\boldsymbol{v}$ má všechny prvky nulové s vyjímkou i-tého prvku, který je jedničkový.

Nejlepší představu o markovském náhodném procesu bez absorpčních stavů dává jeho graf přechodů. Představíme si „značku“, která označuje okamžitý stav systému a která se tedy „pohybuje“ po grafu. V každém stavu, do kterého se dostane, setrvá náhodnou dobu s exponenciálním rozdělením s parametrem λ_i. Parametr má význam výsledné (tj. součtové) intenzity odchodů z odpovídajícího stavu. Střední doba T_i setrvání značky v i-tém uzlu je dána převrácenou hodnotou $1/\lambda_i$, jak již bylo uvedeno. Z i-tého uzlu značka přechází náhodně některou výstupní hranou do dalšího uzlu. Pravděpodobnosti větvení jsou dány poměrem intenzit přechodů výstupních hran. Protože z každého uzlu grafu přechodů vedou cesty do všech ostatních uzlů, projde za dostatečně dlouhý časový interval značka každým uzlem (a setrvá v něm nějakou dobu). Limitní pravděpodobnosti všech stavů tedy budou mít nenulovou hodnotu.

Vzhledem k cyklickému charakteru náhodného procesu můžeme zavést další důležité veličiny. Nepodmíněná střední frekvence f_{ij} přechodů po obecné hraně grafu je průměrný počet přechodů po hraně ze stavu i do j za jednotku času. Je dána vztahem

$$f_{ij} = p_i \lambda_{ij}, \tag{2.8}$$

kde λ_{ij} je intenzita přechodů (podmíněná stř. frekvence přechodů) po uvažované hraně grafu a p_i je limitní pravděpodobnost výchozího stavu hrany.

Střední frekvenci f_i průchodů stavem i získáme součtem frekvencí přechodů po výstupních hranách stavu i jako

$$f_i = p_i \lambda_i = \frac{p_i}{T_i}, \tag{2.9}$$

kde λ_i je součet intenzit přechodů na výstupních hranách stavu i a T_i je střední doba setrvání ve stavu i.

Střední doba T_{ci} cyklu průchodů stavem i, tedy střední doba od jednoho příchodu do dalšího příchodu, je dána převrácenou hodnotou frekvence f_i

$$T_{ci} = \frac{1}{f_i} \tag{2.10}$$

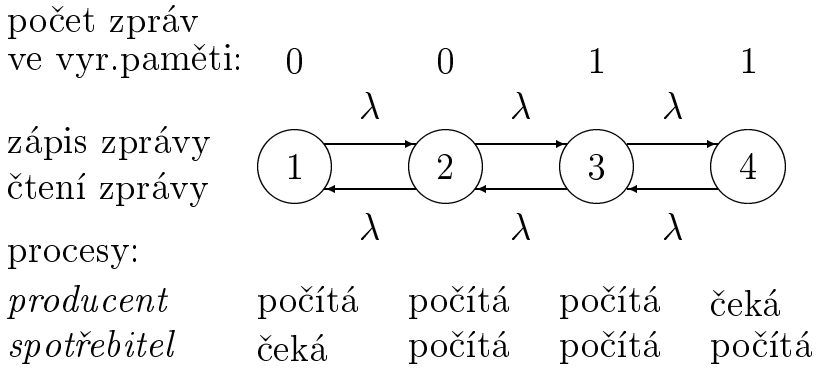
Na frekvenčním principu je rovněž možné provést přímou formulaci rovnic pro výpočet limitních pravděpodobností stavů. Pro každý uzel i se zřejmě musí rovnat střední frekvence příchodů a odchodů ze stavu. V této souvislosti se používá název *frekvenční rovnováha*. Zápisem uvedené podmínky pro všechny stavy a doplněním normalizační podmínky dostaneme soustavu typu (2.7), kterou jsme dříve odvodili limitním přechodem z diferenciálních Kolmogorovových rovnic. Frekvenční rovnováha platí i pro libovolný řez v grafu.

Markovské procesy bez absorpčních stavů lze využít například v oblasti spolehlivostních modelů *obnovovaných systémů* (každou poruchu lze nějak opravit, přechody jsou způsobeny událostmi typů *porucha* a *oprava*), modelů *systémů hromadné obsluhy* (viz dále např. M/M/1, M/M/n) nebo modelů spolupracujících paralelních procesů. Základní podmínkou využití markovských náhodných procesů bez absorpčních stavů je opět pro každý stav exponenciální pravděpodobnostní rozdělení časových intervalů mezi časovým bodem příchodu do stavu a vznikem událostí iniciujících jednotlivé přechody ze stavu.

Příklad 2.2

Uvažujme dva výpočetní procesy spolupracující technikou *asynchronní komunikace* (tj. zasíláním zpráv přes vyrovnávací paměť). Proces *producent*

cyklicky provádí (lokální) výpočet s náhodnou dobou trvání. Tato doba má exponenciální rozdělení s parametrem λ (tj. střední doba výpočtu je $1/\lambda$). Výsledek výpočtu se odesílá zprávou spolupracujícímu procesu *spotřebitel*. Uvažujme nulovou (zanedbatelnou) dobu operace odeslání zprávy. Proces *spotřebitel* zprávu vybere z vyrovnávací paměti (uvažujme nulovou dobu operace čtení zprávy), zprávu zpracuje (opět uvažujeme lokální výpočet s náhodnou exponenciálně rozloženou dobou trvání, parametr λ) a popsanou činnost stále opakuje. Předpokládejme například vyrovnávací paměť s délkou $l = 1$ (tj. pro jednu zprávu). Vzhledem k tomu, že občas dojde k (nechtěnému) zdržení procesů (*producent* čeká nad plnou pamětí nebo *spotřebitel* nad prázdnou) dojde k degradaci teoreticky dosažitelné rychlosti výpočtu. Tuto rychlost můžeme měřit počtem odeslaných či přijatých zpráv za jednotku času, přičemž zmíněná teoretická rychlost (v tomto případě frekvence) je zřejmě λ. Graf přechodů markovského modelu popsaného abstraktního systému je na obrázku 2.3.



Obrázek 2.3: Graf přechodů modelu *producent–spotřebitel*

Odpovídající soustavu rovnic pro limitní pravděpodobnosti stavů formulujme s využitím principu frekvenční rovnováhy (frekvence příchodů do stavu je stejná jako frekvence odchodů):

$$\begin{array}{lrcrcrcl}
1: & p_2\lambda & & = & p_1\lambda & & & \\
2: & p_1\lambda & + p_3\lambda & = & p_2\lambda & + p_2\lambda & = & 2p_2\lambda \\
3: & p_2\lambda & + p_4\lambda & = & p_3\lambda & + p_3\lambda & = & 2p_2\lambda \\
4: & p_3\lambda & & = & p_4\lambda & & &
\end{array}$$

Je možné se přesvědčit, že každou z rovnic lze získat lineární kombinací tří ostatních. Je tedy třeba libovolnou z nich nahradit normalizační podmínkou.

Uvážíme-li dále, že s ohledem na symetrii grafu musí platit $p_1 = p_4$ a $p_2 = p_3$, redukujeme soustavu do tvaru:

$$\begin{array}{lrcrcl} 1: & p_2 & - & p_1 & = & 0 \\ norm: & 2p_1 & + & 2p_2 & = & 1 \end{array}$$

Výsledné limitní pravděpodobnosti stavů zřejmě budou $p_1 = p_2 = p_3 = p_4 = 0,25$. Výslednou frekvenci výpočtu určíme například jako frekvenci zapisování zpráv do vyrovnávací paměti, tedy jako součet frekvencí všech přechodů modelujících zapsání zprávy:

$$f = (p_1 + p_2 + p_3)\lambda = 0,75\lambda$$

Vlivem omezené délky vyrovnávací paměti klesne rychlost modelovaného výpočtu (měřená frekvencí zapisování zpráv) na tři čtvrtiny teoreticky dosažitelné rychlosti λ.

Čtenáře nepochybně napadne, nakolik nepříliš realistický předpoklad exponenciálního rozložení doby vytvoření/zpracování zprávy ovlivnil výsledek.
Pokud by tato doba nebyla náhodná, dojde po počátečním přechodném ději k synchronizaci běhu procesů tak, že výpočet poběží teoreticky dosažitelnou frekvencí λ.
Pokud bude mít náhodná doba vytvoření/zpracování zprávy například rovnoměrné nebo normální rozdělení (tj. „méně náhodné“ než exponenciální), dojde opět k částečné synchronizaci chodu obou uvažovaných procesů a pro frekvenci výpočtu bude platit $0,75\lambda < f < \lambda$. Markovský model tedy v tomto případě (a podobně v mnoha jiných) poslouží k odhadu nejhorší možnosti (angl. worst case analysis).

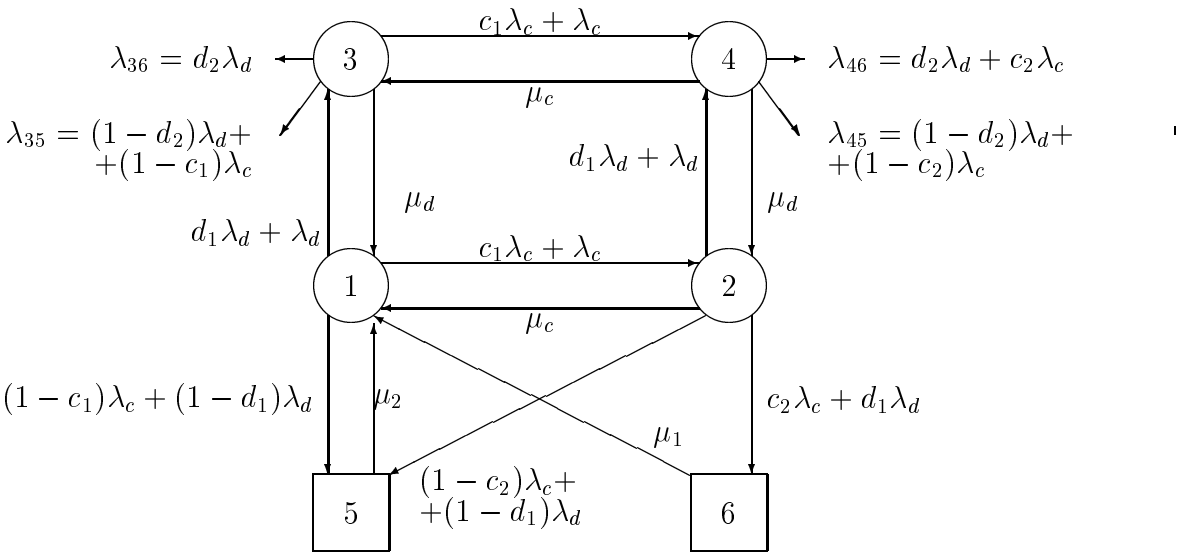
Příklad 2.3

V tomto příkladu předvedeme ukázku markovského spolehlivostního modelu *duplexního* (tj. zdvojeného) výpočetního systému, který obsluhuje důležitou databázi.

První procesor je aktivní a realizuje všechny probíhající transakce. Druhý procesor je využit jako tzv. *horká záloha*. Při poruše aktivního procesoru je na záložním procesoru aktivován záložní výpočetní proces, který s využitím

informace uložené na vnější paměti (angl. *check-point*) dokončí rozpracovanou transakci. Pro uložení databáze je použita zdvojená disková vnější paměť. První disk je aktivní a druhý slouží opět jako horká záloha. Čtení se provádí z aktivního disku, zapisovaná informace je ukládána na oba disky. Při poruše aktivní diskové jednotky je možné dokončit rozpracovanou transakci s využitím informace uložené na záložním disku.

Předpokládáme *nedokonalé pokrytí* poruch procesoru i diskové jednotky. Nepokrytá porucha aktivního prvku (neúspěšná rekonfigurace) vede do stavu, ve kterém je porušena konzistence databáze. Pokrytá porucha vede buď do stavu, ve kterém systém pokračuje v normální činnosti, nebo do stavu, ve kterém systém nepracuje, ale databáze není porušena. Pravděpodobnost pokrytí poruchy bývá označována jako *koeficient pokrytí poruchy*.



Obrázek 2.4: Graf přechodů modelu duplexního databázového systému

Uvažujeme následující parametry úlohy :

$c_1,\ d_1$	koeficient pokrytí poruchy aktivního prvku (procesoru, disku)
$c_2,\ d_2$	koeficient pokrytí druhé poruchy (první dosud neopravena)
λ_c	intenzita poruch procesoru
λ_d	intenzita poruch disku
μ_c	intenzita opravy procesoru (za provozu zbytku systému)
μ_d	intenzita opravy disku (za provozu zbytku systému)
μ_1	intenzita opravy po přerušení činnosti systému

(oprava hardware a restart celého systému)

μ_2 intenzita opravy po porušení databáze
(oprava hardware, restart systému, oprava databáze ze záložní kopie a souboru změn)

Graf přechodů odpovídajícího markovského spolehlivostního modelu je uveden na obrázku 2.4. Stavy modelu mají dále uvedený význam.

1 systém v provozu, všechny prvky pracují normálně
2 systém v provozu, jeden procesor porouchaný
3 systém v provozu, jeden disk porouchaný
4 systém v provozu, jeden procesor a jeden disk porouchaný
5 porucha systému s porušením databáze
6 přerušení činnosti systému, databáze neporušena

Výpočet limitních pravděpodobností stavů by se provedl pro dané hodnoty parametrů dříve uvedeným postupem.

V této souvislosti je vhodné připomenout, že získání věrohodných parametrů modelu je často pracnější než vlastní konstrukce a numerické řešení modelu.

Jako cílový ukazatel (výsledek) se u spolehlivostních modelů obnovovaných systémů nejčastěji určuje tzv. *stacionární koeficient pohotovosti* systému k_p (angl. *availability*), který udává limitní (tj. dlouhodobě měřenou) pravděpodobnost použitelnosti (dostupnosti) systému. V tomto případě zřejmě bude:

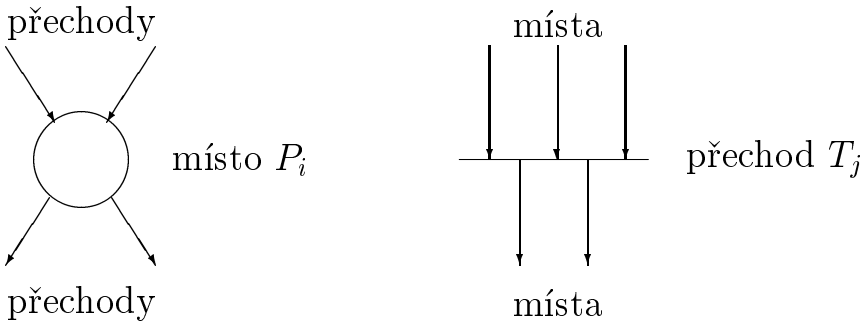
$$k_p = p_1 + p_2 + p_3 + p_4$$

2.2 Petriho sítě

Petriho sítě (dále P-sítě) představují známý a dosti často využívaný formální model diskrétních systémů složených z více aktivních prvků. Účelem dále provedeného popisu není kompletní a matematicky přesný popis aparátu P-sítí, ale spíše jejich konceptuální prezentace s ohledem na možnosti použití k modelování reálných systémů. Pro zájemce o preciznější popis problematiky lze doporučit dobře dostupnou literaturu [Češ94], která na přiložené disketě obsahuje program pro analýzu P-sítí PESIM (pro OS Windows). Základní vlastnosti a aplikace stochastických petriho sítí jsou přehledně popsány například v [PN90].

2.2.1 C/E petriho sítě

Jedná se o základní typ P-sítí využitelný jako model diskrétního systému s *nedefinovanou dynamikou*, tj. s nestanovenými časovými parametry přechodů mezi stavy modelu. P-síť tohoto typu je graf se dvěma typy uzlů označovanými jako *místa* (angl. *places*, popřípadě *podmínky* - angl. *conditions*) a *přechody* (angl. *transitions*, popřípadě *události* - angl. *events*. Označení tohoto typu sítí C/E tedy znamená *Conditions/Events*. Místa a přechody jsou spojeny hranami, které vedou *z míst do přechodů* a *z přechodů do míst* - tedy nikoliv například z míst do míst.



Obrázek 2.5: Místa a přechody

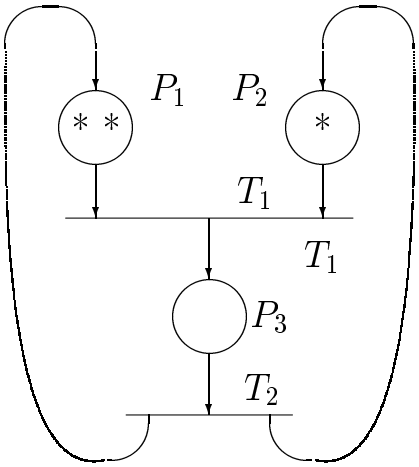
Neformální charakteristiku sítě provedeme následujícím výčtem:

- *Stav místa* je dán jeho *ohodnocením*. Ohodnocení je nezáporné celé číslo, například pro místo P_1 bude ohodnocení m_1.
- *Stav celé sítě* je dán ohodnocením všech míst (celočíselný vektor). Počet stavů sítě není (na rozdíl od konečného automatu) principiálně omezen,

v aplikacích ale zpravidla použijeme síť s konečným počtem možných ohodnocení.

- Pro analýzu sítě potřebujeme znát *počáteční ohodnocení sítě*, tj. počáteční ohodnocení všech míst.
- Ke změně ohodnocení (stavu) sítě dojde realizací některého přechodu. *K přechodu může dojít tehdy, je-li ohodnocení všech vstupních míst přechodu nenulové.*
- Síť je *nedeterministická* v tom smyslu, že pro několik připravených přechodů není stanoveno, který se uskuteční. *Realizace přechodu změní ohodnocení sítě tak, že ohodnocení všech vstupních míst přechodu se dekrementuje (tj. původní minus jedna), naopak ohodnocení všech výstupních míst přechodu se inkrementuje.*

Jako příklad uvedeme C/E-síť, která modeluje dva paralelní výpočetní procesy využívající k interakci *kritickou sekci* (tj. sdílí například společná data). Oba procesy stále cyklicky střídají lokální výpočet, vstup do kritické sekce, výpočet v kritické sekci (jen jeden proces), výstup z kritické sekce.



T_1 přechod – vstup do krit.sekce

T_2 přechod – uvolnění krit.sekce

P_1 místo – ohodnocení m_1 znamená pč. procesů v lok. činnosti

P_2 místo – ohodnocení m_2 znamená pč. volných míst v krit. sekci

P_3 místo – ohodnocení m_3 znamená pč. procesů v krit. sekci

Obrázek 2.6: Model paralelních procesů s kritickou sekcí

V obrázku je naznačeno počáteční ohodnocení sítě (tj. 210). *Je třeba zdůraznit, že jedna petriho síť může sloužit jako model pro větší množství případů* - zde například pro tři procesy z nichž nejvýše dva mohou vstoupit do kritické sekce by se pouze změnilo počáteční ohodnocení sítě na 320.

Zvolený příklad dokumentuje v aplikacích nejčastěji využívaný typ C/E sítě, která má následující vlastnosti:

- Je *konečná* (omezená), tj. má konečný počet možných ohodnocení (zde ohodnocení 210 a 101).
- Je *živá*, tj. každé ohodnocení sítě je (po konečném počtu přechodů) dosažitelné z každého jiného. Živá síť nemá žádný *mrtvý stav*, tj. takové ohodnocení sítě, pro které není realizovatelný žádný přechod.

Uvedený typ sítě se využívá zejména k modelování souběžných vzájemně vázaných cyklických činností (například paralelní procesy, komunikační protokoly, vázané automaty ap.), u kterých hrozí *zablokování* (též *uvíznutí*, angl. *deadlock*). Analýza tohoto typu sítě pak spočívá zejména v určení *množiny dosažitelných ohodnocení sítě* a ověření, že síť je živá.

Existuje řada zobecnění tohoto základního typu sítí, například místa s omezenou kapacitou, hrany se zadanou průchodností, inhibiční hrany (nenulové ohodnocení vstupního místa hrany blokuje přechod) ap. Tato vylepšení jsou vesměs zaváděna s cílem vylepšit modelovací schopnost sítě, tj. umožnit lepší a jednodušší vyjádření reálného problému pomocí P-sítě.

2.2.2 Stochastické petriho sítě

Jak již bylo řečeno, C/E sítě neumožňují modelovat dynamiku systému, tj. uvažovat nějakou dobu trvání modelovaných dějů. Nejjednodušším způsobem, jak do P-sítí zavést *modelový čas* (tj. uvažovat časový vývoj ohodnocení sítě), je stanovit dobu trvání jednotlivých přechodů (tj. dobu od umožnění přechodu nenulovým nastavením vstupních míst do realizace přechodu). Tento časový parametr přechodu může být buď deterministická veličina nebo náhodná veličina s daným pravděpodobnostním rozdělením. Hodnota časového parametru přechodu (nebo parametrů pravděpodobnostního rozdělení doby přechodu) může nebo nemusí záviset na aktuálním ohodnocení sítě.

Dále popíšeme základní využívaný typ stochastických P-sítí označovaný jako GSPN (angl. *Generalized Stochastic Petri Nets*) [PN90]. Tento typ sítí může sloužit jako formální model cyklických (stále se opakujících) souběžných činností, jejichž jednotlivé fáze mají náhodnou dobu trvání (opět například spolupracující paralelní procesy, komunikační protokoly, obnovované systémy se zýšenou spolehlivostí).

GSPN mají zachovány všechny výše uvedené vlastnosti C/E sítí. Mají ale zavedeny dva typy přechodů:

- *Okamžité* (angl. *immediate*) přechody mají v modelovém čase dobu přechodu *rovnou nule* (tj. deterministickou).
- *Časované* přechody mají náhodnou dobu přechodu *s exponenciálním pravděpodobnostním rozdělením*, přičemž parametr rozdělení může být stanoven jako lineární funkce ohodnocení vstupních míst přechodu, nebo může být konstantní. Náhodná doba trvání přechodu (tj. doba od vzniku ohodnocení umožňujícího přechod do realizace přechodu) se obrazně označuje jako *doba hoření přechodu* - angl. *firing time*.

V důsledku dvou kvalitativně odlišných možností přechodů dostaneme i dvojí kvalitativně rozdílná ohodnocení sítě:

- *Nestabilní* ohodnocení umožňuje vznik pouze okamžitých přechodů. Toto ohodnocení trvá nulovou dobu v modelovém čase (tj. v témže modelovém čase se přechází do jiného ohodnocení). Pokud je ohodnocením umožněno několik okamžitých přechodů, není stanoveno, který nastane (nedeterminismus stejný jako u C/E sítí).
- *Stabilní* ohodnocení aktivuje (v přenesené terminologii *zapaluje*) pouze časované přechody. Síť tedy v tomto ohodnocení (stavu) setrvá náhodnou dobu různou od nuly. Pokud ohodnocení zapaluje několik časovaných přechodů, v konkrétním případě (pokusu) dojde k přechodu, který měl nejmenší *dobu hoření*.

Omezíme-li se pouze na konečné a živé sítě, je možné (automaticky, tj. programem) převést GSPN na odpovídající (matematicky *izomorfní*) markovský náhodný proces bez absorpčních stavů. *Izomorfismus* mezi GSPN a markovským náhodným procesem můžeme neformálně charakterizovat zhruba takto:

- Každému stabilnímu ohodnocení sítě odpovídá stav markovského procesu.
- Každému přechodu zapálenému ve stabilním ohodnocení sítě odpovídá v grafu markovského procesu hrana vedoucí ze stavu odpovídajícího tomuto (výchozímu) ohodnocení do stavu odpovídajícího cílovému *stabilnímu* ohodnocení sítě (tj. opět *přechod* v markovském procesu).

- Váha hrany v grafu markovského procesu (tj. intenzita λ_{ij}) se v obecném případě stanoví výpočtem z ohodnocení odpovídajícího výchozímu stavu hrany (lineární kombinace ohodnocení vstupních míst přechodů). Jedná se o parametr příslušného exponenciálního rozdělení doby trvání přechodu.

- Limitní pravděpodobnosti stavů markovského procesu jsou shodné jako limitní pravděpodobnosti odpovídajících ohodnocení v GSPN.

- Střední frekvence výskytu odpovídajících si událostí (např. průchod stavem) jsou přenositelné z markovského procesu do GSPN.

Hrubá filosofie využití GSPN jako modelovacího prostředku je pak následující:

- Jako zdrojový formální model použijeme GSPN, protože má větší modelovací schopnost (též *expresivnost*) než markovský proces. Struktura GSPN je totiž semanticky bližší modelovaným problémům. Dále pomocí GSPN můžeme jednoduše reprezentovat systémy s velkým počtem stavů a pomocí jedné GSPN můžeme reprezentovat velké množství markovských modelů (odlišených počátečním ohodnocením sítě).

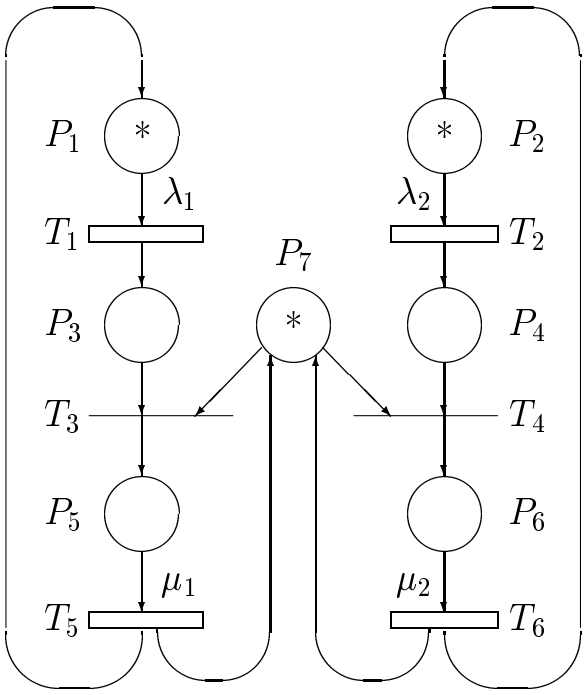
- Pro zadané počáteční ohodnocení GSPN dokážeme automaticky (viz např. program PESIM [1]) převést síť na markovský proces.

- Markovský proces řešíme obvyklým postupem - základní operací je řešení (velké) soustavy lineárních algebraických rovnic pro limitní pravděpodobnosti stavů.

- Získané numerické hodnoty limitních pravděpodobností stavů interpretujeme jako hodnoty limitních pravděpodobností stabilních ohodnocení GSPN. Pro tento účel je třeba v procesu převodu GSPN na izomorfní markovský model nějakým způsobem registrovat konverzní funkci mezi kódem stavu markovského procesu a kódem odpovídajícího ohodnocení v GSPN.

Dále uvedeme jednoduché příklady GSPN. V obrázcích jsou okamžité přechody znázorněny jednoduchou čarou (jako přechody u C/E sítí), časované přechody jsou znázorněny úzkým obdélníčkem.

[1] PESIM umožňuje zadat jen konstantní parametr časovaného přechodu



Obrázek 2.7: Model paralelních procesů s kritickou sekcí

Příklad 2.4

Uvažujeme dva paralelní procesy spolupracující programovací technikou *sdílení dat*. Sdílená data jsou před konfliktním přístupem chráněna mechanismem kritické sekce (jeden *semafor* nebo *paměťový zámek*). Procesy cyklicky opakují *lokální výpočet* a *výpočet v kritické sekci*. Doby příslušných výpočtů mají exponenciální pravděpodobnostní rozdělení, parametry příslušných rozdělení označíme λ_1, μ_1 pro první proces a λ_2, μ_2 pro druhý proces.

Použitá GSPN s počátečním ohodnocením 1100001 je na obrázku 2.7. Obrázek dobře dokumentuje semantickou blízkost modelovaného problému a GSPN. V GSPN jsou zachyceny (jedničkovým ohodnocením příslušného místa) všechny možné stavy procesů - například pro první proces ohodnocení místa P_1 odpovídá lokální činnosti procesu, ohodnocení místa P_3 odpovídá stavu čekání na vstup do kritické sekce a ohodnocení místa P_5 činnosti procesu v kritické sekci. Podobně přechody odpovídají událostem ovlivňujícím stavy procesů, například T_6 znamená uvolnění kritické sekce druhým procesem. Kritická sekce je explicitně modelována místem P_7, ohod-

nocení místa odpovídá počtu procesů, které mohou vstoupit do kritické sekce. V uvedené GSPN jsou zřejmě zahrnuty i (shodné) vývojové diagramy činnosti obou procesů.

V rámci cvičení se pokuste zkonstruovat izomorfní markovský model a z limitních pravděpodobností stavů formulovat vztah pro zmenšení rychlosti výpočtu (vlivem konfliktů na kritické sekci). Využijte analogii s příkladem 2.2. Dále je možné využít uvedený příklad k meditaci nad možností modelování paralelních procesů se složitější strukturu interakcí (více fází činnosti procesu, synchronizace, semafory, bariéry, randez-vous ap.)

Příklad 2.5

Uvažujme obecnější model komunikace paralelních procesů typu *producent – spotřebitel* než model z příkladu 2.2. Zobecnění bude spočívat v tom, že procesů–*producentů* bude r, procesů–spotřebitelů bude s a budou komunikovat přes společnou vyrovnávací paměť (schránku) pro k zpráv. Vytvoření zprávy trvá náhodnou dobu s exponenciálním rozdělením s parametrem (intenzitou) λ, podobně zpracování zprávy má parametr μ.

Použitá GSPN přechodů je na obr. 2.8. Počáteční nastavení sítě je $rs00k0$. Pro časované přechody jsou uvedeny funkce pro výpočet intenzity přechodu v závislosti na ohodnocení vstupních míst přechodu. Význam ohodnocení jednotlivých míst je následující:

P_1 ... Počet procesů typu *producent* připravujících zprávu,

P_2 ... počet procesů typu *spotřebitel* zpracovávajících zprávu,

P_3 ... počet procesů typu *producent* čekajících na uvolnění místa ve vyrovnávací paměti,

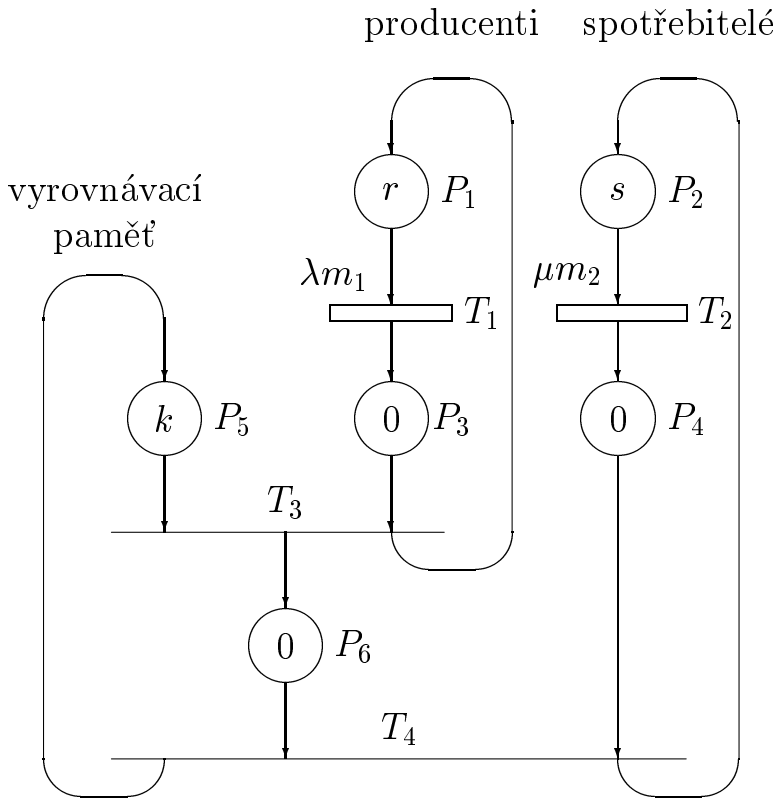
P_4 ... počet procesů typu *spotřebitel* čekajících na vložení zprávy do vyrovnávací paměti,

P_5 ... počet volných míst ve vyrovnávací paměti,

P_6 ... počet zpráv ve vyrovnávací paměti.

Přechody mají tento význam:

T_1 ... Žádost procesu o zápis zprávy do vyrovnávací paměti,



Obrázek 2.8: Model komunikace *producenti–spotřebitelé*

T_2 ... žádost procesu o čtení zprávy z vyrovnávací paměti,

T_3 ... zápis zprávy do vyrovnávací paměti,

T_4 ... čtení zprávy z vyrovnávací paměti.

Ještě naznačíme využití sítě k získání konkrétních výsledků. Předpokládejme, že nás zajímá střední frekvence zapisování zpráv do vyrovnávací paměti (měřítko rychlosti výpočtu, největší možná střední frekvence zapisování je λr). Zápis zprávy je modelován přechodem sítě z každého ohodnocení $m_1 x 0 x m_5 x$, kde m_1 a m_5 jsou nenulová ohodnocení míst P_1 a P_5 a na ohodnocení dalších míst nezáleží (symbol x) [2]. Střední podmíněná frekvence každého takového přechodu je λm_1 (tj. je závislá jen na ohodnocení místa P_1). Po převodu sítě na izomorfní markovský model je třeba určit limitní pravděpodobnosti všech stavů odpovídajících množině ohodnocení $m_1 x 0 x m_5 x$. Dále se určí nepodmíněné střední frekvence přechodů

[2] Uvažujeme jen stabilní ohodnocení iniciující přechod - přes ohodnocení s nenulovou hodnotou m_3 se přejde s nulovým zpožděním.

jako $p_i \lambda m_1$, kde p_i je limitní pravděpodobnost příslušného ohodnocení sítě. Výslednou frekvenci zapisování zpráv získáme jako jako součet nepodmíněných frekvencí přechodu pro všechna ohodnocení z množiny $m_1 x 0 x m_5 x$.

V analogii s tímto příkladem sestrojte znovu GSPN z příkladu 2.4. pro případ, že oba procesy mají stejné parametry λ, μ.

Kapitola 3

Systémy hromadné obsluhy

Jako *systémy hromadné obsluhy* (dále zkráceně SHO) jsou označovány více či méně formální modely reálných systémů, jejichž funkce spočívá v realizaci *obsluhy* (tj. poskytnutí nějaké *služby*) pro velké počty průběžně přicházejících *požadavků* (též *transakcí*). Služby poskytují prvky označované jako *kanály obsluhy* či *servery*, před kanálem obsluhy ze zpravidla vytváří *fronta* požadavků čekajících na poskytnutí služby. *Významnou vlastností SHO je abstrakce od semantiky poskytované služby, modeluje se pouze časová posloupnost přicházejících požadavků a jejich časová náročnost na poskytované služby. Vzhledem k velkým počtům požadavků je přiměřený pravděpodobnostní popis časových charakteristik proudů požadavků.* Model typu SHO je vytvářen s cílem určit výkonnostní charakteristiky systému, například *střední dobu od zadání do obsloužení požadavku, střední délku některé fronty požadavků* ap. V anglosaské literatuře je místo pojmu SHO používán termín *queuing network* (tj. *síť front*, *queue* jako fronta), disciplina označovaná jako *queuing theory* se zabývá matematickou analýzou sítí front a obslužných uzlů.

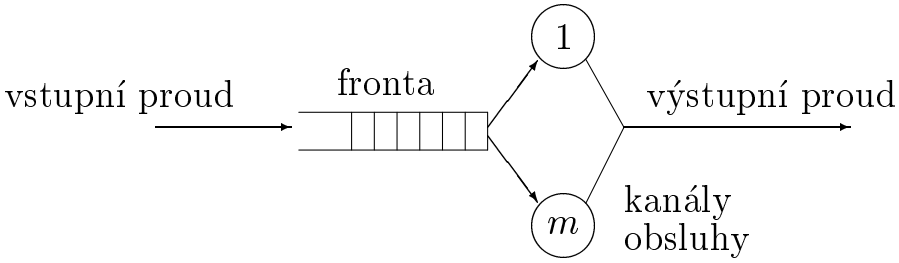
Jako příklad reálného systému modelovaného prostřednictvím SHO může sloužit například počítačový systém, dopravní síť (*požadavek* je abstrakcí projíždějícího vozidla, *kanál obsluhy* je abstrakcí např. křižovatky) nebo velký úřad (*požadavek* je abstrakcí procházejícího občana, *kanál obsluhy* je abstrakcí úředníka realizujícího příslušný úřední úkon, *frontu* v tomto kontextu si jistě každý dovede představit).

Při dostatečném zjednodušení vlastností reálného systému lze SHO konstruovat jako *matematický* model poskytující požadované výsledky *v uzavřeném tvaru* (tj. jako vzorec). Detailnější modelování vlastností vyžaduje zpravidla koncipovat SHO jako *simulační* model. Následující přehled SHO

zahrnuje základní používané matematické modely, problematika simulačního modelování je předmětem kapitoly 4.

3.1 Elementární SHO

Konceptuální model elementárního obslužného systému je naznačen na následujícím obrázku 3.1 Jako *stav* systému figuruje celkový počet požadavků akumulovaných v systému (tj. ve frontě i v obslužných kanálech).



Obrázek 3.1: Elementární SHO

Dále kvalitativně popíšeme jednotlivé prvky obslužného systému, uvedeme veličiny využívané k jejich matematickému popisu a rovněž uvedeme základní používanou klasifikaci obslužných systémů.

Základním předpokladem pro další popis je *stacionární režim činnosti systému*, tj. časová stálost zdrojových statistických charakteristik (parametrů) modelovaného systému (modelujeme *ustálený provoz*, nikoliv *přechodný* děj).

3.1.1 Zdroj požadavků

Zdroj požadavků může být buď *omezený* nebo *neomezený*. U omezeného zdroje je populace zákazníků produkujících požadavky omezená (konečná) a *vstupní proud požadavků je závislý na stavu obslužného systému* (tj. zmenšuje se s narůstající délkou fronty). U neomezeného zdroje vstupní proud nezávisí na stavu SHO.

3.1.2 Vstupní proud požadavků

Předpokládáme, že požadavky vstupují do SHO *náhodně* v časové posloupnosti $\{t_0 < t_1 < t_2 \ldots\}$. Náhodná veličina $\tau_k = t_k - t_{k-1}$ (kde $k \geq 1$)

je nazývána *interval příchodů* (angl. *interarrival time*). Předpokládáme-li statistickou nezávislost a stejné pravděpodobnostní rozdělení jednotlivých hodnot τ_k, lze je považovat za realizace jedné náhodné veličiny τ.

Nejčastěji využívaným způsobem matematického popisu vstupního proudu je zadání distribuční funkce pravděpodobnostního rozdělení $F_a(t) = \mathcal{P}\{\tau \leq t\}$ *nebo odpovídající hustoty pravděpodobnosti* $f_a(t)$ (derivace distribuční funkce) [1]. Nejčastěji využívaným typem vstupního proudu je tzv. *poissonovský vstupní proud*, ve kterém má interval příchodů *exponenciální pravděpodobnostní rozdělení*. Vstupy požadavků přicházejících v poissonovském proudu jsou *nahodilé* v tom smyslu, že pro náhodně vybírané stejně velké (ale velmi malé ve srovnání se střední dobou intervalu příchodů) časové intervaly je stále stejná pravděpodobnost příchodu požadavku.

Dále uvedeme veličiny a vztahy charakterizující vstupní proud:

$F_a(t)$ – distribuční funkce pravděpodobnostního rozdělení časového intervalu mezi příchody požadavků. Základní charakteristika vstupního proudu.

$E\{\tau\} = T_a = 1/\lambda$ – střední hodnota intervalu mezi příchody (střední *perioda* příchodů). Veličina λ zřejmě představuje střední *frekvenci* příchodů požadavků.

$F_a(t) = 1 - e^{-\lambda t}$ – distribuční funkce časových intervalů mezi příchody v poissonovském proudu. Střední frekvence příchodů λ je (jediným) parametrem rozdělení. *poissonovský proud tedy lze plně charakterizovat zadáním jediné číselné hodnoty.*

$C_a = \sigma\{\tau\}/T_a$, kde C_a je tzv. *koeficient variance* a $\sigma\{\tau\}$ je směrodatná odchylka intervalu mezi příchody. Koeficient variance číselně charakterizuje *nahodilost* (resp. *pravidelnost*) příchodů, pro zcela *pravidelné* (tj. nenáhodné) příchody má hodnotu 0, pro příchody v poissonovském proudu má hodnotu 1, pro většinu reálných případů hodnotu mezi 0 a 1. Pro vstupní proud obsahující *shluky* požadavků může být hodnota koeficientu variance větší než 1.

3.1.3 Fronta požadavků

Je charakterizována maximálním počtem míst ve frontě (tj.*maximální délkou*) a *frontovou disciplinou*, tj. pravidlem, podle kterého se vybírá z fronty

[1] Aproximaci hustoty pravděpodobnosti lze z reálného systému experimentálně určit konstrukcí histogramu - viz dále.

požadavek k obsluze. Délka může být *omezená* (i na nulu - například u požadavků na spojení v telefonní ústředně) nebo *neomezená*, druhý uvedený případ se používá jako abstrakce vedoucí zpravidla k zjednodušení matematického řešení SHO. Frontová disciplina může být buď *bez priorit* nebo *prioritní* (tj. uvažují se priority požadavků). Prioritní discipliny mohou být *s přednostním vyloučením* (též *preempcí*) nebo bez přednostního vyloučení. Přednostní vyloučení znamená přerušení probíhající obsluhy při příchodu požadavku s vyšší prioritou a používá se zejména tehdy, je-li možné obsluhu snadno přerušit a opět obnovit bez ztráty již vykonané činnosti (ve víceuživatelském operačním systému bude fronta procesů připravených k výpočtu s přednostním vyloučením, kdežto fronta souborů připravených k tisku musí být bez přednostního vyloučení). Nejčastěji využívanou frontovou disciplinou je FIFO (*First In First Out*) nebo FIFO s prioritami. Za zmínku stojí dále fronty *se sdílením času*, kde všechny požadavky se cyklicky střídají v obsluze s pevnou nebo proměnlivou hodnotou tzv. *časového kvanta* obsluhy.

K popisu fronty se dále zavádí tyto veličiny:

w – okamžitý počet požadavků ve frontě (náhodná funkce času),

$E\{w\} = L_w$ – střední počet požadavků ve frontě, tj. střední délka fronty,

t_w – doba čekání ve frontě pro jeden konkrétní požadavek (náhodná veličina),

$E\{t_w\} = T_w$ – střední doba čekání požadavků ve frontě.

3.1.4 Kanály obsluhy

Počet kanálů obsluhy označíme jako m, nejčastějším případem je jednokanálová obsluha, tj. $m = 1$. V SHO je doba t_s obsluhy konkrétního požadavku obvykle chápána jako náhodná veličina s daným pravděpodobnostním rozdělením (stejným pro všechny požadavky) [2]. Základní charakteristikou kanálu tedy opět bude distribuční funkce uvažovaného rozdělení, tj. $F_s(t) = \mathcal{P}\{t_s \leq t\}$ [3].

Dále uvedeme veličiny a vztahy charakterizující jeden kanál obsluhy:

[2] Pro konkrétní proud požadavků v konkrétním reálném systému lze získat aproximaci hustoty rozdělení konstrukcí histogramu.

[3] Tato charakteristika zahrnuje jak vlastní *výkonnost* kanálu, tak specifiku nějakého proudu požadavků (např. jeho *rozptyl* doby obsluhy). Vyměníme-li procesor počítače za

$F_s(t)$ – distribuční funkce pravděpodobnostního rozdělení doby obsluhy. Základní charakteristika kanálu obsluhy.

$E\{\tau\} = T_s = 1/\mu$ – střední doba obsluhy. Veličina μ zřejmě představuje střední *frekvenci* obsluh, tj. střední počet požadavků obsloužených za jednotku času *za předpokladu, že je kanál stále zatížený.*

$F_s(t) = 1 - e^{-\mu t}$ – distribuční funkce exponenciálního rozdělení doby obsluhy. Střední frekvence obsluh μ je (jediným) parametrem rozdělení.

$C_s = \sigma\{t_s\}/T_s$, kde C_s je koeficient variance doby obsluhy a $\sigma\{t_s\}$ je směrodatná odchylka doby obsluhy. Koeficient variance číselně charakterizuje *nahodilost* (resp. *pravidelnost*) obsluh. Pro shodné (tj. konstantní, nenáhodné) doby obsluh má hodnotu 0, pro obsluhy s exponenciálním rozdělením má hodnotu 1, pro většinu reálných případů hodnotu mezi 0 a 1 [4].

3.1.5 Kendallova klasifikace elementárních SHO

Tato klasifikace v základní podobě využívá pro elemenární SHO symbolické označení $A/B/m$, kde za A se dosazuje symbolický typ pravděpodobnostního rozdělení vstupních intervalů a za B symbolický typ pravděpodobnostního rozdělení doby obsluhy. Symbol m jsme již zavedli pro označení počtu (identických) kanálů obsluhy v elementárním SHO. Základní symboly používané k označení typu pravděpodobnostního rozdělení jsou:

GI – Pro vstupní proud se statisticky nezávislými intervaly mezi příchody a obecným (tj. jakýmkoliv) pravděpodobnostním rozdělením těchto intervalů (GI jako *general independent*).

G – Pro obecné (tj. jakoukoliv funkci $F_s(t)$) rozdělení doby obsluhy [5].

M – Pro exponenciální rozdělení doby obsluhy nebo intervalu příchodů (M jako *markovský* případ).

dvakrát *výkonnější* (měřeno např. v MIPS), znamená to změnu původně uvažovaného (resp. experimentálně zjištěného) rozdělení $F_s(t)$ na $F_s(t/2)$ (jako kdyby se dvakrát zrychlil čas).

[4] Měřením reálných proudů požadavků v počítačových systémech lze zjistit i případy s $C_s > 1$. Takovéto proudy lze modelovat *hyperexponenciálním* rozdělením (náhodná směs několika exponenciálních rozdělení) nebo diskrétním rozdělením.

[5] Implicitně se předpokládá statistická nezávislost hodnot doby obsluhy v časové posloupnosti požadavků. Na reálném proudu požadavků lze tento předpoklad experimentálně ověřit výpočty korelačních koeficientů (autokorelace).

D – Pro deterministické (konstantní, pravidelné) intervaly příchodů nebo doby obsluhy.

Někdy se ještě doplňují údaje o maximální délce fronty a frontové disciplině. Například symbolické označení

$$M/D/1/\infty/FIFO$$

představuje elementární SHO s exponenciálním rozdělením intervalu příchodů, stejnou (tj. konstantní, nenáhodnou) dobou obsluhy pro všechny požadavky, jedním kanálem obsluhy, neomezeným počtem míst ve frontě a FIFO frontovou disciplinou. (poslední dva údaje představují obvyklé implicitní hodnoty, takže stačí uvést jen $M/D/1$).

3.1.6 Veličiny a vztahy pro elementární SHO jako celek

V analogii s již dříve zavedenými veličinami pro kvantitativní popis fronty zavedeme obdobné veličiny pro elementární SHO jako celek:

q – okamžitý celkový počet požadavků v SHO, tj. ve frontě i v kanálech obsluhy. Jedná se o (časově proměnnou) náhodnou veličinu kvantifikující *stav* SHO.

$E\{q\} = L_q$ – střední celkový počet požadavků v SHO,

t_q – doba průchodu celým SHO pro jeden konkrétní požadavek (náhodná veličina), je též označována jako *doba odezvy.*

$E\{t_q\} = T_q$ – střední doba průchodu požadavků elementárním SHO (*střední doba odezvy*).

Pravděpodobně nejvýznamnější veličinou ovlivňující chování SHO je *zatížení* (též *časové využití* nebo *koeficient využití*) obslužných kanálů [6].

$$\varrho = \frac{1}{m}\frac{T_s}{T_a} = \frac{1}{m}\frac{\lambda}{\mu} \tag{3.1}$$

[6]Předpokládáme, že obslužné kanály jsou identické a že jsou zatěžovány rovnoměrně. Rovnoměrné zatížení kanálů lze zajistit například pravidlem, že volný kanál vlastní aktivitou sleduje stav fronty a přebírá (bez zpoždění) obsluhu prvního požadavku ve frontě (je-li nějaký) - příkladem může být klasický obchod s jednou frontou zákazníků a s několika prodavačkami za pultem.

Nutnou podmínkou pro dosažení *stacionárního režimu* v činnosti SHO je hodnota $\varrho < 1$. Pokud tato nerovnost není splněna, narůstá v čase neomezeně délka fronty (SHO se "zahltí" přicházejícími požadavky).

Hodnota například $\varrho = 0,5$ znamená:

- Kanály obsluhují nějaký požadavek pouze polovinu (dostatečně dlouhé) doby sledování, tj. jejich zatížení je *poloviční*,
- časové využití (všech) kanálů je 50%,
- pravděpodobnost obsazení kanálu (realizuje obsluhu) v náhodně vybraném okamžiku je $0,5$,
- střední počet požadavků vyskytujících se v kanálech obsluhy je v tomto případě $L_s = 0,5m$ (veličina analogická k L_w a L_q).

Veličina $u = \lambda/\mu$ se nazývá *intenzita provozu* a k ní nejbližší vyšší celé číslo představuje minimální počet obslužných kanálů potřebných (pro dané T_a a T_s) pro zajištění stacionárního režimu činnosti SHO.

Pro elementární SHO dále zřejmě platí:

$$\begin{aligned} L_q &= L_w + L_s = L_w + m\frac{\lambda}{\mu} \\ T_q &= T_w + T_s = T_w + \frac{1}{\mu} \end{aligned} \tag{3.2}$$

Veličiny L_q, T_q a L_w, T_w jsou vzájemně jednoduše nahraditelné díky existenci tzv. *Littleových vzorců*, které platí pro všechny elementární SHO *s neomezeným počtem míst ve frontě*:

$$L_q = \lambda T_q, \quad L_w = \lambda T_w \tag{3.3}$$

Littleovy vztahy jsou intuitivně dobře pochopitelné na základě úvahy, že za dobu T_q průchodu *průměrného* požadavku přes SHO přijde *v průměru* λT_q dalších požadavků a stejný počet je obsloužen (tj. odejde), takže *v průměru* setrvává v SHO $L_q = \lambda T_q$ požadavků.

V uvedených rovnicích 3.2 a 3.3 lze brát λ, μ, m jako součást zdrojového popisu SHO (tj. parametry), zbývající veličiny L_q, L_w, T_q, T_w jsou zpravidla předmětem matematického řešení SHO (tj. výsledky). Důležité je, že stačí určit jednu (kteroukoliv) z nich, a ostatní tři se jednoduše vypočítají z 3.2 a 3.3.

3.1.7 Výstupní proud požadavků

Zde ještě uvedeme, jak se změní proud požadavků (přesněji statistické charakteristiky časových intervalů mezi následujícími požadavky v proudu) po průchodu obslužným systémem. Předně předpokládáme stacionární režim v činnosti SHO (tj. $\varrho < 1$) [7]. V tomto režimu všechny požadavky, které vstoupí do SHO jsou v konečném čase obslouženy a *střední frekvence i perioda odchodů požadavků je shodná se střední periodou a frekvencí příchodů* (tj. T_a a λ). *Nahodilost* (resp. *pravidelnost*) odchodů je pro malé hodnoty zatížení určována charakterem pravděpodobnostního rozdělení intervalů mezi příchody (tj. rozdělení intervalů mezi odchody se blíží rozdělení $F_a(t)$), pro velké hodnoty zatížení je určující distribuční funkce rozdělení doby obsluhy $F_s(t)$.

Významný je poznatek, že ve stacionárním režimu poissonovský vstupní proud po průchodu obslužným systémem s exponenciálním rozdělením doby obsluhy si zachová poissonovský charakter (tj. pravděpodobnostní rozdělení intervalů mezi příchody i mezi odchody je exponenciální, navíc se stejným parametrem λ).

Uvedené kvalitativní závislosti lze *přibližně* kvantifikovat s využitím vztahu (Allen) popisujícího změnu koeficientu variance náhodného časového intervalu po sobě jdoucích požadavků v proudu (míra *pravidelnosti* proudu) po průchodu SHO:

$$C_o^2 \doteq 1 + \varrho^2(C_s^2 - 1) + (1 - \varrho^2)(C_a^2 - 1) \tag{3.4}$$

V uvedeném vzorci je C_o koeficient variance výstupního proudu, C_a je koeficient variance vstupního proudu, ϱ je zatížení a C_s je koeficient variance doby obsluhy.

[7] Pro nestacionární režim ($\varrho \geq 1$) je zřejmě střední frekvence výstupního proudu rovna $m\mu$ (obslužné kanály stále pracují).

3.2 Jednokanálové SHO

V této podkapitole uvedeme bez odvození základní analytické výsledky pro nejběžnější typy elementárních SHO. Soustředíme se na vysvětlení a intuitivní pochopení těchto vztahů. *Jako vstupní data pro analytické řešení elementárních SHO slouží funkce pravděpodobnostních rozdělení $F_a(t)$ a $F_s(t)$.* V dále uváděných vzorcích figuruje jako hlavní parametr zatížení ϱ, jehož číselnou hodnotu lze určit ze zadaných funkcí $F_a(t)$ a $F_s(t)$ výpočtem středních hodnot a dosazením do 3.1. *Cílem řešení je v obecném případě určení funkcí pravděpodobnostních rozdělení veličin w, q, t_w a t_q, nebo alespoň jejich středních hodnot L_w, L_q, T_w a T_q.* Jak již bylo řečeno, stačí určit kteroukoliv z těchto čtyř středních hodnot, ostatní jsou závislé.

3.2.1 M/M/1

Základní elementární SHO, který slouží jako porovnávací případ pro jiné. Vstupní proud požadavků je poissonovský, proud lze charakterizovat jediným parametrem λ (střední frekvence proudu a zároveň parametr exponenciálního rozdělení $F_a(t) = 1 - e^{-\lambda t}$). Doba obsluh má exponenciální rozdělení $F_s(t) = 1 - e^{-\mu t}$, kde (jediný) parametr μ má zároveň význam střední (dosažitelné) frekvence obsluh [8]. Konkrétní SHO typu $M/M/1$ lze zadat dvěma hodnotami (reálnými čísly) parametrů λ a μ. Zatížení SHO je potom $\varrho = \lambda/\mu$, a pro dosažení stacionárního režimu činnosti musí být menší než 1.

Dále uvedeme překvapivě jednoduchý vztah pro střední hodnotu počtu požadavků akumulovaných v SHO L_q [9], veličiny L_w, T_q a T_w se pokuste cvičně odvodit sami z 3.2 a 3.3.

$$L_q = \frac{\varrho}{1-\varrho}, \quad \varrho < 1 \tag{3.5}$$

Ještě uvedeme, že náhodná doba t_q strávená požadavkem v SHO má opět exponenciální pravděpodobnostní rozdělení se střední hodnotou T_q a připomeneme, že *výstupní proud požadavků z M/M/1 má opět poissonovský*

[8] Pokud má SHO zatížení $\varrho < 1$ (stacionární režim), je zřejmě skutečná střední frekvence obsluh rovna střední frekvenci vstupního proudu λ, je tedy menší než μ.

[9] Odvození se provede určením limitních pravděpodobností stavů markovského náhodného procesu bez absorpčních stavů (viz odst.2.1.2), stav procesu odpovídá stavu SHO, tj. veličině l_q. O odvození se z cvičných důvodů pokuste sami, limitní pravděpodobnost stavu k (tj. stavu, ve kterém je v SHO akumulováno k požadavků) vychází $p_k = \varrho^k(1-\varrho)$.

charakter.

Příklad 3.1

Do počítačového systému určeného například k rezervaci letenek přichází proud požadavků na rezervace. Střední frekvence proudu požadavků je $\lambda = 10\ sec^{-1}$, tj. v průměru přijde každých $T_a = 0,1\ sec$ jeden požadavek. Vzhledem k tomu, že systém sbírá nezávislé požadavky z mnoha míst, lze předpokládat poissonovský charakter proudu požadavků (tj. požadavky přichází *nahodile*, *pravděpodobnost příchodu dalšího požadavku v každé milisekundě je stejná, zde tedy* $0,01$ [10]. Požadavky se vyřizují postupně jak přichází (FIFO fronta) a bez přerušení (uzamkne se celá databáze). Průměrná doba obsluhy jednoho požadavku je $T_s = 0,08\ sec$, v proudu jsou náhodně smíchány požadavky s velmi krátkou i relativně dlouhou dobou obsluhy - *lze tedy dobu obsluhy považovat za náhodnou veličinu s exponenciálním rozdělením.* Parametr tohoto rozdělení je $\mu = 1/T_s$.

Při výpočtu postupujeme například takto:

- Nejprve určíme zatížení systému jako $\varrho = \lambda/\mu = \lambda T_s = 0,8$. Hodnota zatížení je menší než jedna a modelovaný systém tudíž bude pracovat ve stacionárním režimu.

- Střední počet požadavků akumulovaných v SHO je podle (3.5) $L_q = \varrho/(1-\varrho) = 0,8/0,2 = 4$.

- Střední dobu odezvy určíme podle Littleova vzorce (3.3) jako $T_q = L_q/\lambda = 0,4\ sec$. [11]

- Střední délka fronty požadavků čekajících na vyřízení bude $L_w = L_q - L_s = L_q - \varrho = 3,2$.

[10] Přibližně, tj. pokud zanedbáme možnost příchodu dvou či více požadavků v téže milisekundě.

[11] V reálném systému by byla doba odezvy horší o komunikační zpoždění mezi pracovní stanicí zadávající požadavek a serverem databáze se záznamy rezervací.

Uvedený příklad dokumentuje skutečnost, že při snaze o co nejvyšší využití kanálu obsluhy (tj. při $\varrho \to 1$) se výrazně (hyperbolická závislost (3.5)) zhorší ukazatele L_w a T_q, tj. narůstá střední délka fronty a střední doba odezvy. *Například pro počítače realizující aplikace v reálném čase je určující požadovaná (krátká) doba odezvy a zatížení systému je třeba přizpůsobit (tj. snížit)* [a].

[a] Manažerské závěry vyplývající pro obchod s jednou prodavačkou (model M/M/1) se pokuste cvičně odvodit sami.

3.2.2 M/G/1

Předpoklad poissonovského vstupního proudu požadavků je dostatečně realistický pro velký počet aplikací SHO. Naproti tomu exponenciální rozdělení doby obsluhy často ani zhruba neodpovídá vlastnostem reálného proudu požadavků. Proto modelem M/G/1 (symbol G znamená obecné rozdělení doby obsluhy dané nějakou distribuční funkcí $F_s(t)$) v mnoha případech lépe popíšeme chování reálného systému. *Jako zdrojové parametry modelu M/G/1 typicky slouží střední frekvence λ vstupního proudu požadavků a funkce $F_s(t)$ (nebo hustota $f_s(t)$) popisující pravděpodobnostní rozdělení doby obsluhy.* Zatížení je možné z parametrů stanovit jako $\varrho = \lambda T_s$, kde T_s je střední hodnota rozdělení $F_s(t)$.

Dále uvedeme bez odvození vztah pro střední délku fronty L_w v systému M/G/1. Tento vztah je jedním z nejdůležitějších vzorců v matematické teorii systémů hromadné obsluhy.

$$L_w = \frac{\varrho^2}{2(1-\varrho)}(1 + C_s^2) \tag{3.6}$$

Symbol C_s označuje koeficient variance doby obsluhy (viz dříve v souvislosti s popisem kanálu obsluhy). Připomínáme, že další veličiny (T_w, L_q, T_q) lze stanovit prostřednictvím (3.2) a (3.3).

Koeficient C_s kvantifikuje nahodilost (resp. pravidelnost) doby obsluhy:

- Pro exponenciální rozdělení doby obsluhy (tj. M/G/1 $\rightarrow$ M/M/1) je $C_s = 1$ a vztah (3.6) se zjednoduší do podoby platné pro M/M/1, tedy $L_w = \varrho^2/(1-\varrho)$.
- Pro shodné (nenáhodné) doby obsluhy (tj. M/G/1 $\rightarrow$ M/D/1) je $C_s = 0$ *a střední délka fronty* L_w *vyjde dvakrát menší než pro M/M/1 se stejným zatížením.*
- Pro "nepravidelné" doby obsluhy (popsané příkladně gaussovským rozdělením s danou střední hodnotou a směrodatnou odchylkou) je $0 < C_s < 1$ a střední délka fronty L_w vychází někde mezi výše uvedenými krajními případy.
- Pro případy, kdy $C_s > 1$ (hyperexponenciální nebo diskrétní rozdělení doby obsluhy), může vyjít střední délka fronty L_w "horší" než pro M/M/1.

Příklad 3.2

Uvažujme, že v počítačovém systému pro rezervaci letenek z příkladu 3.1 trvá obsluha každého požadavku stejně dlouho. Systém *bude stejně zatížen* jako v příkladu 3.1, tedy konstantní (nenáhodná) doba obsluhy je $T_s = 0,08$ *sec*. Zřejmě se jedná o případ označený v Kendallově klasifikaci M/D/1, se zatížením $\varrho = 0,8$.

Střední délka fronty je $L_w = \varrho^2/2(1-\varrho) = 0,64/0,4 = 1,6$, tedy poloviční proti dříve uvedené hodnotě z příkladu 3.1. Střední počet požadavků akumulovaných v SHO lehce získáme jako $L_q = L_w + \varrho = 2,4$ a střední doba odezvy bude $T_q = L_q/\lambda = 2,4/10 = 0,24$ *sec*.

3.2.3 GI/G/1

Pro případ elementárního SHO se vstupním proudem požadavků se statisticky nezávislými intervaly příchodů a obecným rozdělením velikosti intervalu (symbol GI) musíme znát kromě rozdělení doby obsluhy $F_s(t)$ ještě rozdělení intervalů příchodů, tj. $F_a(t)$ nebo $f_a(t)$. Zatížení je pak možné

stanovit jako $\varrho = T_s/T_a$, kde T_s a T_a jsou střední hodnoty obou uvedených rozdělení.

Střední délku fronty L_w lze *přibližně* stanovit náhradou jedničky v závorce ve vztahu (3.6) druhou mocninou koeficientu variance časových intervalů požadavků ve vstupním proudu (tj. původně $C_a^2 = 1$ pro M/G/1):

$$L_w = \frac{\varrho^2}{2(1-\varrho)}(C_a^2 + C_s^2) \tag{3.7}$$

Střední délka fronty (a doba odezvy) závisí jak na pravidelnosti příchodů, tak na pravidelnosti obsluh. Pro příchody s pevnou (nenáhodnou) periodou a shodné doby obsluhy všech požadavků (tedy případ D/D/1, $T_a > T_s$) z uvedeného vzorce správně vychází nulová délka fronty (každý požadavek je obsloužen dříve než přijde další).

3.3 Otevřené sítě front

Komplikovanější reálné obslužné systémy (např. počítačová síť, dopravní síť, finanční úřad) zpravidla obsahují větší počet kanálů obsluhy (serverů), každý s vlastní frontou požadavků na obsloužení. Požadavky vstupují do systému (např. poplatník do finančního úřadu), přechází (i cyklicky) mezi jednotlivými servery a po ukončení všech dílčích obsluh vystupují ze systému. Takovýto systém lze zřejmě modelovat jako síť, jejímiž prvky jsou elementární SHO. Sítě s explicitními vstupy z okolí se nazývají *otevřené*. Na tomto místě se budeme zabývat matematickým modelem [12] sítě front.

Otevřenou síť front můžeme znázornit orientovaným grafem, jehož uzly odpovídají elementárním SHO a hrany popisují možnosti přechodu požadavků mezi jednotlivými uzly. Počet elementárních SHO v síti označíme jako n, číslování (index i) odpovídajících uzlů tedy bude v intervalu $< 1, n >$. Jako váhy hran se obvykle využívají tzv. *pravděpodobnosti větvení*, tj. hrana vedoucí z uzlu i do uzlu j je vážena pravděpodobností p_{ij} přechodu do j po ukončení obsluhy v i. Pochopitelně součet vah všech hran vycházejících z uzlu i musí být roven jedné. Okolí obslužného systému (resp. *zdroj požadavků*) lze modelovat jako zvláštní uzel (s indexem například $i = 0$), hrany

[12] Simulační modely sítí front budou zmíněny později. Připomínáme, že matematický model (za cenu většího či menšího zjednodušení modelované skutečnosti) lze verifikovat a navíc často poskytuje řešení v *uzavřeném tvaru* (tj. jako vzorec či vzorce zobrazující prostor parametrů modelu do prostoru výsledků).

s váhami p_{0j} pak modelují vstupy požadavků z okolí do j–tého serveru a hrany s váhami p_{i0} modelují odchod požadavku ze sítě po ukončení i–té dílčí obsluhy.

Opět předpokládáme *stacionární režim činnosti* modelovaného systému, tj. *předpokládáme, že parametry vstupních toků požadavků se v průběhu "života" systému nemění (neuvažujeme např. pondělní ranní špičku na finančním úřadě) a dále že všechny elementární SHO v síti pracují se zatížením menším než jedna.*

3.3.1 Analýza středních frekvencí a zatížení

Budeme rozlišovat vnitřní toky požadavků jednotlivými uzly sítě a toky požadavků mezi uzly sítě. Střední frekvenci vnitřního toku uzlem i označíme jako Λ_i. Frekvence Λ_0 patří celkovému toku požadavků vstupujících z okolí do sítě front. Rozdělují–li se požadavky po ukončení obsluhy v i–tém uzlu zcela náhodně podle pravděpodobností větvení p_{ij}, bude zřejmě střední frekvence toku požadavků z uzlu i do j dána jako $\lambda_{ij} = \Lambda_i p_{ij}$. Na vstupu i–tého uzlu se sčítají frekvence toků procházejících po hranách vedoucích do uzlu. Ve stacionárním režimu činnosti sítě *nedochází k hromadění požadavků v žádném uzlu sítě* a pro obecný uzel i uvažované sítě musí platit:

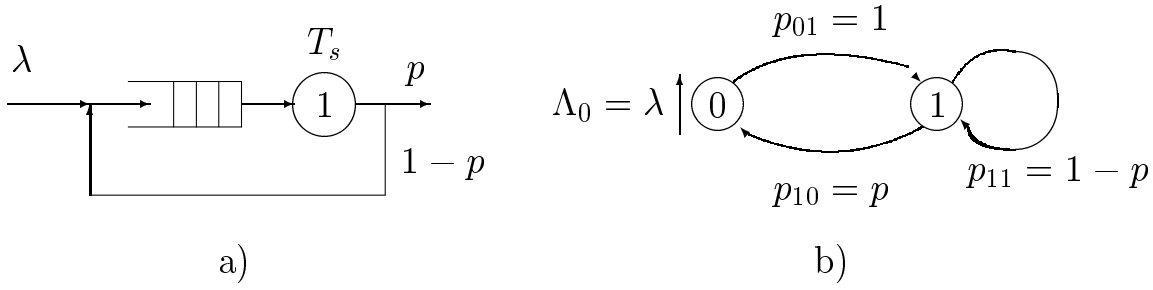
$$\sum_k \Lambda_k p_{ki} = \Lambda_i = \sum_j \Lambda_i p_{ij} \tag{3.8}$$

kde index k nabývá hodnot odpovídajících vstupním hranám a index j výstupním hranám uzlu i. Jedná se o jakousi *rovnici kontinuity* toku procházejícího uzlem, tj. součet frekvencí vstupujících toků je roven frekvenci vnitřního toku uzlem a zároveň součtu frekvencí všech vystupujících toků.

Pro určení hodnot středních frekvencí všech toků v síti musí být dána hodnota frekvence Λ_0 (tj. střední frekvence celkového toku vstupujícího do sítě) a matice $\{p_{ij}\}$ pravděpodobností větvení (tj. čtvercová matice s rozměrem $n+1$). Hodnoty frekvencí Λ_i lze získat řešením soustavy n lineárních algebraických rovnic sestavených pro uzly 1 až n podle (3.8). Po výpočtu středních frekvencí vnitřních toků lze provést kontrolu předpokladu stacionárního režimu činnosti jednotlivých uzlů výpočtem jejich zatížení. Pro tento výpočet potřebujeme pro obecný uzel i znát ještě počet m_i kanálů obsluhy a střední dobu obsluhy T_{si}. Zatížení je pak dáno jako $\varrho_i = (1/m_i)\Lambda_i T_{si}$ a musí vyjít menší než jedna pro všechny uzly.

Příklad 3.3

Uvažujme finanční úřad odbavující daňová přiznání. Uvažujeme stacionární režim práce úřadu (zjednodušující předpoklad) při kterém přichází s vyplněným daňovým přiznáním v průměru 15 poplatníků za hodinu, tj. střední frekvence příchodu požadavků do obslužného systému je $\lambda = 15\ hod^{-1}$. Odbavení jednoho přiznání trvá v průměru 3 min, tedy střední doba obsluhy $T_s = 0,05\ hod$. Z deseti vyřizovaných přiznání je v průměru jedno vyplněno špatně, poplatník provede opravu (zjednodušeně uvažujme, že oprava je časově velmi krátká) a znovu se vrací do fronty na obsluhu. Pravděpodobnost vyřízení přiznání v prvním průchodu je tedy $p = 0,9$, pravděpodobnost návratu do obsluhy je $1 - p = 0,1$. Naznačenou situaci modelujeme jednoduchou sítí (jen jeden elementární SHO, $n = 1$) front podle obrázku 3.2a. Znázornění sítě grafem je provedeno na obr. 3.2b.



Obrázek 3.2: Model finančního úřadu

Vnitřní tok Λ_1 obslužným uzlem určíme řešením následující rovnice:

$$\Lambda_1 = \lambda + \Lambda_1(1 - p) \tag{3.9}$$

Řešení je zřejmě $\Lambda_1 = \lambda/p = 15/0,9 = 50/3 = 16,66\ hod^{-1}$. Nárůst frekvence způsobený opakovanými průchody obsluhou je tedy $1,66\ hod^{-1}$. Průměrný počet průchodů jednoho požadavku obsluhou bude $\Lambda_1/\lambda = 1/p = 1,111$. Zatížení obslužného kanálu (tj. úředníka odbavujícího přiznání) bude $\varrho_1 = \Lambda_1 T_s = 16,66\ .\ 0,05 = 0,833$, tedy menší než jedna (z osmihodinové pracovní doby zůstane více než hodina na kávu a jiná odlehčení).

3.3.2 Určení délky front a doby odezvy

Prozatím jsme neuvažovali pravidelnost či nepravidelnost příchodů požadavků do obslužného systému a rozptyl dob jejich obsluhy (tj. příslušná

pravděpodobnostní rozdělení). *Střední frekvence toků požadavků a zatížení obslužných uzlů na tvaru pravděpodobnostních rozdělení nezávisí* (jsou ovlivněny jen středními hodnotami příslušných rozdělení). Naproti tomu střední délky front a doby odezvy narůstají s *nepravidelností* intervalů mezi příchody a dob obsluhy.

Jednoduché matematické řešení sítě front je možné jenom při splnění následujících předpokladů (tzv. *Jacksonův teorém*):

- Všechny toky požadavků z okolí do sítě mají poissonovský charakter.
- Všechny obslužné uzly mají exponenciální rozdělení doby obsluhy se střední hodnotou T_{si}.
- Po ukončení obsluhy v uzlu i přechází požadavek zcela náhodně do dalšího uzlu j (tj. s pravděpodobností p_{ij}), přitom přechod se uskuteční bez zpoždění.

Při splnění uvedených předpokladů jsou všechny toky v síti poissonovské (sloučení dvou či více poissonovských toků rezultuje opět na poissonovský tok, náhodné větvení poissonovského toku vytvoří dílčí poissonovské toky a průchodem poissonovského toku kanálem s exponenciálním rozdělením doby obsluhy zůstane zachován poissonovský charakter toku). *Každý uzel sítě je pak možné uvažovat odděleně jako M/M/1 (resp. M/M/m) elementární SHO s frekvencí vstupního toku Λ_i a střední dobou obsluhy T_{si}.* Jednotlivé uzly se řeší samostatně podle vzorců pro M/M/1, čímž získáme hodnoty veličin L_{qi}, L_{wi}, T_{qi} a T_{wi}.

Pro celou síť je pak možné určit průměrný počet požadavků L_q akumulovaných v celé síti a střední dobu T_q průchodu požadavku sítí jako:

$$L_q = \sum_{i=1}^{n} L_{qi}, \quad T_q = \frac{1}{\Lambda_0} L_q \tag{3.10}$$

Pokud nejsou uvedené předpoklady splněny a některé příchody či obsluhy jsou *pravidelnější* než exponenciální (tj. koeficienty variance odpovídajících rozdělení jsou menší než jedna), lze využít uvedený postup výpočtu k získání jednostranného odhadu, který dává horší hodnoty (větší délky front, delší doby odezvy) než ve skutečnosti nastanou (analýza nejhoršího případu, angl. *worst case analysis*).

Příklad 3.4

Pro model finančního úřadu z příkladu 3.3 předpokládejme dále příchody poplatníků v poissonovském proudu (realistický předpoklad) a exponenciální rozdělení doby obsluhy (předpokládá velký rozptyl – tj. vyřízení některých přiznání trvá relativně dlouho, naopak jiná přiznání jsou vyřízena (nebo vrácena k opravě) obratem). Parametrem exponenciálního rozdělení intervalů ve vstupním proudu je střední frekvence příchodů ($\lambda = 15\ hod^{-1}$), parametrem rozdělení doby obsluhy je střední (dosažitelná) frekvence obsluh, tedy $\mu = 1/T_s = 20\ hod^{-1}$. Zatížení kanálu obsluhy jsme určili v příkladu 3.3 jako $\varrho = 0,833$.

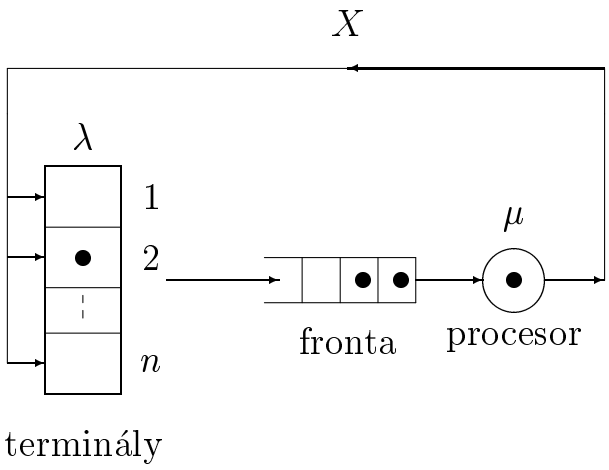
Střední počet požadavků akumulovaných v síti (tj. poplatníků vyskytujících se v modelovaném finančním úřadu) je potom $L_q = \varrho/(1-\varrho) = 0,833/0,166 \doteq 5,018$. Střední délka fronty je $L_w = L_q - \varrho \doteq 4,185$. Střední dobu strávenou poplatníkem v úřadu určíme z Littleova vzorce jako $T_q = L_q/\lambda = 5,018/15 = 0,334\ hod \doteq 20\ min$. *Pokud bude doba obsluhy pravidelnější než by odpovídalo exponenciálnímu rozdělení (např. všichni mají podobné a řádně vyplněné přiznání), lze předpokládat zmenšení střední délky fronty a střední doby odezvy pod uvedené hodnoty.*

3.4 Uzavřené sítě front

Uzavřené sítě front se liší od předchozí kategorie tím, že nemají žádné explicitní vstupy požadavků z okolí. *V síti se "pohybuje" pevná populace požadavků, které modelují nějakou aktivitu s neomezenou dobou trvání* (na rozdíl od otevřených sítí, kde aktivita modelovaná průchodem požadavku sítí je časově ohraničená). Sítě tohoto typu nacházejí uplatnění zejména při modelování výpočetních systémů s omezenou množinou (stále aktivních) uživatelů. Na rozdíl od předchozího typu sítí analýzou modelu určujeme typicky střední frekvence průchodů požadavků určitým místem sítě, tyto frekvence pak vedou k určení výkonnostních ukazatelů s charakterem *propustnosti* (též *průchodnosti*), tj. středního počtu nějakých operací vykonávaných za jednotku času. Relativně jednoduché analytické řešení je opět možné jen v případech, kdy doby obsluhy (či jiné doby) mají exponenciální pravděpodobnostní rozdělení. V dalších dvou odstavcích stručně uvedeme klasické modely *interaktivního terminálového systému* a *multiprogramního výpočetního systému s dávkovým režimem práce.*

3.4.1 Model interaktivního výpočetního systému

Model interaktivního výpočetního systému se skládá z jednoho elementárního SHO modelujícího vlastní počítač (včetně OS). V nejjednodušším případě uvažujeme *jeden kanál obsluhy* (procesor), FIFO frontu a exponenciální pravděpodobnostní rozdělení doby obsluhy s parametrem μ. K počítači je připojeno n terminálů, které jsou stále v provozu. Terminál je abstrakcí uživatele systému – předpokládá se, že všichni uživatelé se u terminálu chovají stejně, tj. nezáleží na tom, zda se uživatelé u terminálu střídají či nikoliv. Parametrem terminálu je náhodná doba, po kterou uživatel čeká mezi splněním jednoho jeho příkazu (např. spuštění programu, výpis adresáře, jedna akce v editoru; splnění příkazu se projeví ohlášením (*prompt*) na obrazovku) a zadáním dalšího. Doba je označována jako *doba přemýšlení* (angl. *thinking time*) a budeme zde předpokládat, že má exponenciální pravděpodobnostní rozdělení s parametrem λ. Požadavkem (v dosud užívaném smyslu) je zde průběžná aktivita uživatele terminálu, která se zřejmě může nacházet ve třech stavech – přemýšlení před zadáním příkazu (terminál), čekání (fronta) na zpracování zadaného příkazu, vlastní zpracování příkazu (obsluha) [13]. Model je schematicky znázorněn na obr. 3.3.



Obrázek 3.3: Model interaktivního výpočetního systému

Stav systému může být například definován jako počet požadavků vy-

[13] Modelují se pouze doby vykonávání příkazů (a doby čekání), semantika příkazů (jak je při použití konceptu SHO zvykem) se nemodeluje. Při využití simulačního modelu můžeme v případě potřeby modelovat libovolné detaily, mj. semantiku operací výkonávaných jednotlivými servery.

skytujících se v obslužném SHO (tj. aktivity uživatelů ve stavech *fronta* a *obsluha*). Takto zavedený stav může mít celkem $n+1$ hodnot. Limitní pravděpodobnosti stavů označíme p_0, p_1, ..., p_n a předpokládáme, že je umíme určit [14]. Hodnota například $p_1 = 0,1$ znamená, že 10% z doby života modelovaného systému je tento ve stavu, kdy počítač zpracovává jeden konkrétní příkaz, žádný další příkaz není připraven ke zpracování a u $n-1$ terminálů sedí uživatelé a přemýšlí, co vlastně chtějí zadat.

Ze známých limitních pravděpodobností stavů umíme určit střední frekvenci vykonávání příkazů – je označována jako *propustnost* X. Podmíněná střední frekvence vykonávání příkazů (je-li stále co vykonávat) je zřejmě μ, nepodmíněná (bereme–li v úvahu všechny možné stavy) frekvence je:

$$X = \mu(1 - p_0)$$

Dále lze jednoduše určit *dobu odezvy* T_q, kterou zde definujeme jako střední dobu vykonání jednoho příkazu [15]. Střední doba "obrátky" jednoho konkrétního z n vykonávaných požadavků zřejmě bude n/X [16]. Vzhledem k tomu, že jedna obrátka kromě doby odezvy obsahuje ještě dobu přemýšlení před zadáním dalšího příkazu, dostaneme pro dobu odezvy následující vztah:

$$T_q = \frac{n}{X} - \frac{1}{\lambda} \tag{3.11}$$

3.4.2 Model multiprogramního výpočetního systému

Tento model je též označován jako *model počítače s centrální obsluhou* (angl. *central server model*). V modelu jsou elementární SHO modelující procesor (centrální obsluha) a řadiče periferií (např. vnější paměť, tiskárna ap.). Jednotlivé obslužné subsystémy jsou spojeny do uzavřené sítě front. V síti

[14] Limitní pravděpodobnosti lze získat konstrukcí a řešením matematického markovského modelu bez absorbčních stavů. Model se z cvičných důvodů pokuste sestrojit sami.

[15] Jedná se o důležitý ukazatel sledovaný při návrhu interaktivních výpočetních systémů. Doba odezvy pro jednoduché příkazy by neměla překročit jednu až dvě vteřiny, jinak uživatel ztrácí pozornost (a při zvláště dlouhých odezvách může chaoticky mačkat jiné klávesy než by měl).

[16] Uvedený model není příliš realistický, zejména z toho důvodu, že operační systém obvykle realizuje sdílení času, tj. časově "dlouhé" příkazy se po vyčerpání stanovené (malé) doby přeruší a vrací do fronty. Tím se zlepšuje doba odezvy pro krátké běžné příkazy (viz předchozí poznámka) a prodlužuje se doba vykonání příkazů s charakterem například spuštění časově náročného výpočtu.

"cirkuluje" pevný počet požadavků modelujících chování současně zpracovávaných (multiprogramovaných) zakázek (angl. *job*).

Chování lze ve stručnosti charakterizovat tak, že vykonávaný program nějakou (náhodnou) dobu využívá procesor, potom potřebuje nějakou I/O operaci (s pravděpodobností odpovídající četnosti příslušného typu operace), uvolní procesor a je operačním systémem zařazen do fronty na příslušný řadič periferie. Provedení periferní operace trvá náhodnou dobu (modelována pravděpodobnostním rozdělením doby obsluhy v kanálu modelujícím řadič, střední doba obsluhy odpovídá střední době provedení příslušné I/O operace). Po provedení periferní operace je program operačním systémem opět zařazen do fronty na procesor.

Ukončení výpočtu programu je modelováno odchodem požadavku z procesoru zvláštní větví (tj. nikoliv větví vedoucí do SHO modelujícího některý řadič periferie). Vzhledem k tomu, že při pevném počtu zpracovávaných programů je ukončený program ihned nahražen jiným z dávky zakázek připravené ve vnější paměti, vede zmíněná větev zase zpět do fronty na procesor (nový program začíná využitím procesoru, nikoliv periferní operací). Střední frekvenci průchodů větví překlenující procesor lze tudíž interpretovat jako střední počet zpracovaných programů za jednotku času, tj. *propustnost* X.

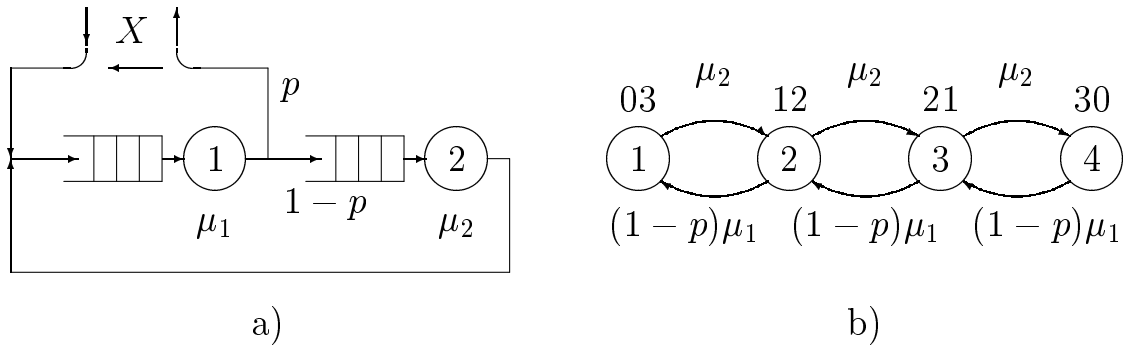
Modelům tohoto typu byla věnována v šedesátých a sedmdesátých letech velká pozornost v souvislosti s optimalizací činnosti operačních systémů sálových počítačů (angl. *mainframe*) pracujících v dávkovém režimu [17].

Příklad 3.4

V tomto příkladu sestrojíme model počítače s multiprogramním operačním systémem a virtuální pamětí. Z periferních zařízení budeme uvažovat jen vnější paměť, jejíž řadič realizuje výměnu stránek mezi operační a vnější pamětí. Schematické znázornění modelu jako uzavřené sítě front je provedeno na obrázku 3.4a.

Kanál obsluhy číslo 1 modeluje procesor, kanál obsluhy číslo 2 modeluje řadič vnější paměti. Stupeň multiprogramování (tj. počet multiprogramovaných zakázek) budiž například $n = 3$. *Předpokládáme, že všechny zpracovávané programy se chovají stejně, a tudíž lze využít jen jednu množinu*

[17] Simulační model výpočetního systému diskutovaného typu je popsán dále v části příkladů využití simulačního nástroje C-Sim.



Obrázek 3.4: Model počítače s virtuální pamětí

parametrů popisujících chování programu - viz veličiny p, μ_1, μ_2. Program počítá (tj. využívá procesor – požadavek modelující chování programu se nachází v prvním kanálu obsluhy) až do přístupového konfliktu (chybí adresovaná stránka v operační paměti), poté je operačním systémem zařazen do fronty na obslužný kanál 2 (ten provádí přenos chybějící stránky z vnější do operační paměti). Pravděpodobnost větvení s hodnotou například $p = 0,1$ znamená, že z deseti případů uvolnění procesoru vykonávaným programem v průměru jedno znamená ukončení programu (a okamžité doplnění dalšího programu do počtu n), v devíti případech program uvolňuje procesor, protože nemá v paměti potřebná data nebo instrukce.

Uvažujme exponenciální pravděpodobnostní rozdělení doby obsluhy v procesoru i v řadiči vnější paměti (parametry μ_1, μ_2). Pro procesor nemusíme být daleko od reality, protože doby mezi přístupovými konflikty mohou mít velký rozptyl. Naproti tomu pro případ řadiče disků by lépe odpovídalo gaussovské nebo rovnoměrné rozdělení doby obsluhy (doba přenosu stránky z vnější do operační paměti trvá vždy stejně dlouho, nepravidelnost vnáší náhodná poloha čtecích hlaviček v okamžiku zahájení obsluhy přenosu konkrétní stránky). Pro exponenciálně rozdělené doby obsluhy ale můžeme sestrojit markovský model (bez absorpčních stavů) znázorněný na obr. 3.4b [18]. Stavy modelu jsou dány kombinacemi možností rozmístění požadavků v obslužných subsystémech (zde čtyři možnosti, v obrázku uvedené kódování například 30 znamená všechny požadavky vyskytující se v obslužném subsystému procesoru a žádný v obslužném subsystému disků). Z modelu je možné určit limitní pravděpodobnosti stavů p_1, p_2, p_3 a p_4. Průchodnost X je

[18] Z cvičných důvodů se pokuste sestrojit matematický model v podobě stochastické Petriho sítě, použitelný pro každou hodnotu n.

možné ze známých limitních pravděpodobností stavů určit jako střední frekvenci průchodů větví označenou X (odpovídající přechody nejsou v modelu 3.4b explicitně vyjádřeny hranami, protože vedou ze stavu do téhož stavu), tedy:

$$X = \mu_1 p\ (p_2 + p_3 + p_4)$$

Rovněž je jednoduše možné určit průměrnou dobu zpracování jedné zakázky (tj. průměrnou dobu, po kterou se požadavek modelující zpracování zakázky pohybuje v síti) jako $T_q = 3/X$.

Kapitola 4

Simulační modely diskrétních stochastických systémů

4.1 Principy experimentálního pravděpodobnostního modelování

Experimentální pravděpodobnostní modelování (v nejjednodušší podobě) spočívá v programové realizaci většího množství pokusů s náhodně generovanými parametry pokusu a v následném statistickém vyhodnocení výsledků množiny pokusů.

Pro numerické techniky založené na generování a zpracování náhodných čísel se často používá název *metody Monte Carlo* podle místa, v němž náhoda hraje velkou roli. Pokusů je třeba k získání dostatečně přesných výsledků provést značné množství (typicky *tisíce* až *desítky tisíc*). Generování náhodných parametrů pokusů se realizuje prostřednictvím programově realizovaných *generátorů náhodných čísel* (viz dále). Pro metody Monte Carlo obecně platí, že nejsou příliš přesné (dosažitelná přesnost je typicky řádu *jednotek procent*) a jsou dosti náročné na čas výpočtu. Jsou dobře paralelizovatelné, protože jednotlivé pokusy (nebo skupiny pokusů) lze zpravidla provádět paralelně a jejich výpočet není třeba synchronizovat.

Metody Monte Carlo lze použít i k výpočtu úloh z principu deterministických, například k výpočtu určitých integrálů (i vícerozměrných) a parciálních diferenciálních rovnic. Přirozenější je jejich využití k modelování (řešení) úloh se stochastickým charakterem. Přitom nejsme nijak zvlášť omezováni charakterem používaných pravděpodobnostních rozdělení (ve většině dříve diskutovaných matematických modelů jsme byli omezeni předpokladem exponenciálního rozdělení) a do modelu můžeme zahrnout vcelku li-

bovolnou (rozumnou) úroveň detailů. *Simulační model založený na experimentálním pravděpodobnostním modelování ale provádí (v rámci jednoho simulačního experimentu) pouze transformaci množiny číselných parametrů modelu na množinu číselných výsledků. Na rozdíl od matematického modelu je třeba pro získání nějakých závislostí výsledků na parametrech realizovat větší množství (časově náročných) experimentů.*

Jako jednoduchý příklad uvažujme nějaký matematický vzorec – třeba $x = \sqrt[3]{a^3 + \sqrt{b}}$. V tomto vzorci jsou a a b náhodné veličiny se zadaným (známým) pravděpodobnostním rozdělením. Zajímá nás pravděpodobnostní rozdělení výsledku x [1]. Postup numerického řešení na bázi pravděpodobnostního modelování bude následující:

- S využitím programových generátorů náhodných čísel vygenerujeme konkrétní hodnoty náhodných veličin a_j, b_j (metody generování náhodných čísel viz dále v 4.2).
- Hodnoty a_j, b_j dosadíme do uvažovaného vzorce a určíme odpovídající hodnotu x_j (výsledek pokusu).
- Celý postup opakujeme N–krát, získané hodnoty x_j například ukládáme do jednorozměrného pole.
- Po ukončení všech N pokusů [2] zapamatované hodnoty statisticky zpracujeme – zde konstrukcí tzv. *histogramu* (numerický výpočet nejběžnějších statistických charakteristik viz dále v 4.3).

Složitější situace nastává, nelze-li jednoduše oddělit jednotlivé pokusy. Příkladem budiž elementární SHO (jedná se o systém vyvíjející se v čase, též *dynamický* systém), kde jako jeden pokus lze chápat průchod jednoho požadavku. Protože požadavků pochopitelně může být v SHO současně větší počet, *musíme začít další pokus dříve, než byl předchozí ukončen.* Toto vede k použití konceptu *diskrétního modelového času*, který slouží jako prostředek pro uspořádané provádění (relace *dříve – později*) jednotlivých *fází* jednotlivých (rozpracovaných) pokusů. Nejběžnější způsoby dekompozice modelu reálného dynamického systému využívajícího techniku experimentál-

[1] Jedná se o úlohu tzv. *citlivostní analýzy*, při které se zjišťuje *citlivost* výsledku na *nepřesnost* parametrů výpočtu.

[2] Dále v 4.3. jsou preferovány zejména způsoby průběžného zpracování pokusů, kdy není třeba plýtvat pamětí při ukládání rozsáhlých množin výsledků.

ního pravděpodobnostního modelování (a dále také odpovídající programovací techniky) se nazývají:

- metoda interpretace událostí,
- metoda (pseudo)paralelních procesů.

Obě metody jsou charakterizovány dále v odst. 4.4.2, přičemž hlavní pozornost je věnována metodě paralelních procesů, která je univerzálnější a vede k dekompozici více přibližující model k realitě. Programovací nástroj **C–Sim** popisovaný v kap. 5 je určen pro realizaci simulačních modelů metodou paralelních procesů.

4.2 Generování náhodných čísel

Pokusy o generování náhodných čísel počítačem se objevily ihned se vznikem prvních číslicových počítačů. Nejprve šlo o rozsáhlé tabulky (posloupnosti) čísel uložené na vnější paměti. Dále se využívaly fyzikální generátory náhodných čísel jako speciální periferie (princip např.: čítače vyzářených částic, vzorkování tepelného šumu odporů). Postupně v praxi převážily programově realizované generátory náhodných čísel, *které z posledního (nebo několika posledních) prvku posloupnosti náhodných čísel počítají podle vhodně zkonstruovaného vzorce další prvek.* Na první pohled se nejedná o žádnou náhodu (proto se také často takto získané posloupnosti označují jako *pseudonáhodné*), nicméně pokud má posloupnost patřičné statistické vlastnosti – zejména *splňuje požadované pravděpodobnostní rozdělení* a *sousední prvky posloupnosti (dvojice, trojice ap.) jsou statisticky nezávislé*, lze ji pro úlohy Monte Carlo využít stejně dobře jako posloupnost skutečně náhodných čísel získávanou například z fyzikálního generátoru. Navíc mají *pseudonáhodné posloupnosti* cennou vlastnost *reprodukovatelnosti*, tj. lze je zcela přesně opakovat při dalším běhu programu [3].

[3] Program realizující metodu Monte Carlo dá tedy při každém běhu *pro stejná vstupní data stejné výsledky.* Tuto skutečnost zvláště oceníme při ladění složitějších modelů realizovaných např. technikou *paralelních procesů* - pokud bychom nepracovali s reprodukovatelnými posloupnostmi náhodných čísel, lokalizace a odstranění *run-time* chyb by bylo úlohou hodnou Sherlocka Holmese.

Programově realizované (knihovní) generátory náhodných čísel zpravidla generují čísla s *rovnoměrným pravděpodobnostním rozdělením* a to buď *reálná čísla s rovnoměrným rozdělením na intervalu* $\langle 0, 1)$ nebo *celá nezáporná čísla generovaná se stejnou pravděpodobností na intervalu daném zobrazením celých čísel v příslušném počítači* (viz odst. 4.2.1). Pokud jsou zapotřebí náhodná čísla s jiným než rovnoměrným rozdělením, vytváří se programově sekundární generátor, který vyvolává základní generátor s rovnoměrným rozdělením a výsledky nějak transformuje na požadované rozdělení. Základní možnosti takovéhoto postupu jsou upřesněny dále v 4.2.2 až 4.2.6

4.2.1 Generování čísel s rovnoměrným rozdělením

Nejčastěji používanými metodami pro generování rovnoměrně rozdělených nezáporných celých náhodných čísel jsou tzv. *kongruentní metody*. Tyto metody jsou založeny na využití vzorců typu

$$y_{j+1} = (C_1 + C_2 y_j)_{mod} M, \tag{4.1}$$

kde y_{j+1} je generovaný prvek náhodné posloupnosti, y_j je poslední (předchozí) prvek posloupnosti, C_1, C_2, M jsou číselné parametry generátoru a binární operátor *mod* realizuje výpočet zbytku po celočíselném dělení obou operandů. Vhodnou volbou hodnot C_1, C_2, M (a také výchozí hodnoty y_0) lze ovlivnit kvalitu (zejména rovnoměrnot a statistickou nezávislost) generované posloupnosti (viz např. [Oleh82], [Hurt82]). Zřejmě největší možné číslo na výstupu generátoru může být $M - 1$. Obvykle se volí $M = 2^n$, kde n je počet bitů, na kterých se v příslušném počítači zobrazuje kladné číslo (bez znaménka).

Z uvedeného vztahu vyplývá, že generovaná posloupnost [4] náhodných čísel je *periodická* (jakmile se objeví hodnota, která se již vyskytla, posloupnost se opakuje). *Perioda* opakování posloupnosti bude potom evidentně menší než M. Kvalitní generátor by měl mít co nejdelší periodu [5] a jeho

[4] Ve statistice se spíše používá pojem *soubor náhodných čísel*, zde slovem *posloupnost* zdůrazňujeme, že prvky souboru mají uspořádání (v případě potřeby používáme index j) dané posloupností volání generátoru, tj. každé volání generátoru vrací jiný (další) prvek ze souboru náhodných čísel s příslušným rozdělením.

[5] Za tím účelem se obvykle kombinuje několik náhodných posloupností - například pomocí jedné (pseudo)náhodně indexujeme v tabulce, najdeme uloženou hodnotu y_j, vypočítáme výstup generátoru y_{j+1} (druhá posloupnost) a přepíšeme pozici v tabulce.

volání (výpočet jedné hodnoty posloupnosti) by mělo být co nejkratší [6] s ohledem na potřebné velké počty náhodných čísel vyplývající z podstaty metod Monte Carlo.

Jako příklad běžně dostupných knihovních procedur pro generování náhodných čísel uvedeme funkce z knihovny `stdlib.h` pro jazyk **C**. Jedná se o funkce:

`int rand (void)` – vrací nezáporná celá čísla v $\langle$ `0`, `RAND_MAX` $\rangle$. Využívá se jako základní generátor pro vytvoření čísel s jakýmkoliv rozdělením (viz metody dále). Výchozí hodnota posloupnosti (výsledek prvního vyvolání) je pevně (implicitně) stanovena.

`void srand (unsigned seed)` – umožňuje nastavit výchozí hodnotu (tzv. *násadu* generátoru) prostřednictvím parametru `seed`.

`int random (int num)` – vrací nezáporná celá čísla se stejnou pravděpodobností od 0 do $num - 1$ (včetně obou mezních hodnot). Využívá se pro rozhodování typu *ano – ne* ($num = 2$), popřípadě k náhodnému určení jedné z několika stejně pravděpodobných možností (např. házení kostkou, $num = 6$).

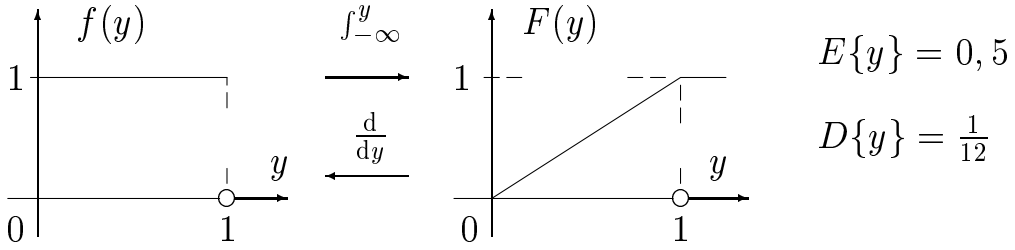
`void randomize (void)` – využívá se k náhodnému (odvozeno od času) počátečnímu nastavení generátoru. *Typicky se používá jen jednou v inicializační sekvenci již odladěné aplikace* (viz předchozí poznámka o obtížném ladění programu při práci se skutečně náhodnými čísly).

V dalším textu budeme uvažovat *normalizované* rovnoměrné pravděpodobnostní rozdělení reálných čísel na $\langle 0, 1)$. *Pokud máme k dispozici generátor celých nezáporných čísel se stejnou pravděpodobností (viz např.* `rand()`*), realizujeme jednoduše transformaci těchto čísel na normalizované rovnoměrné rozdělení konverzí výstupu základního generátoru na reálné číslo a dělením horní mezí intervalu zobrazitelných celých čísel* (viz např. `RAND_MAX`) [7]. Soubor čísel s normalizovaným rozdělením budeme dále označovat symbolem Y na rozdíl od souboru čísel s jiným rozdělením, který označíme X.

[6] Právě kongruentní metody založené na vzorci (4.1) jsou výpočetně efektivní, vyžadují jen jednu dlouhou operaci (násobení), výpočet zbytku lze při výše uvedené volbě N realizovat pomocí posunů.

[7] Ještě je třeba zamezit možnost výskytu hodnoty 1 na výstupu generátoru (zdroj potíží při použití transformační metody).

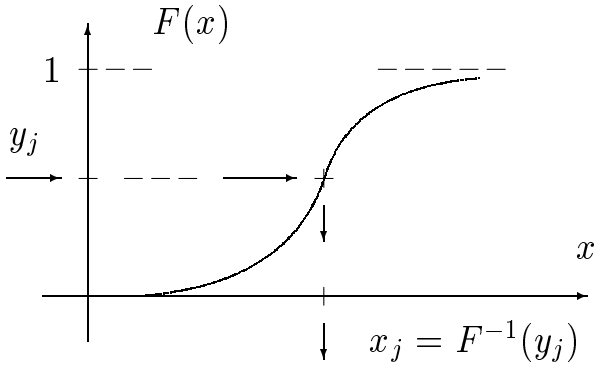
Jeden (blíže neurčený) prvek souboru budeme označovat jako y nebo x (pomocí indexu j rozlišujeme časové uspořádání hodnot x, dále pomocí indexů i, k nějaká jiná uspořádání zřejmá z kontextu). Normalizované rovnoměrné rozdělení je znázorněno na obr. 4.1 [8].



Obrázek 4.1: Normalizované rovnoměrné rozdělení

4.2.2 Metoda inverzní transformace

Jedná se o metodu umožňující *transformaci* náhodných čísel Y s normalizovaným rovnoměrným rozdělením na čísla X zadaná distribuční funkcí $F(x)$ jejich pravděpodobnostního rozdělení. Princip metody inverzní transformace je znázorněn na obrázku 4.2.



Obrázek 4.2: Princip metody inverzní transformace

[8] Rozdělení generované počítačem je ve skutečnosti *kvazirovnoměrné* („děravé“), protože ne všechna reálná čísla v intervalu $\langle 0, 1)$ lze v počítači zobrazit. Pokud ale statistické vlastnosti množiny čísel generovaných počítačem odpovídají statistickým vlastnostem náhodného výběru z ideální (spojité) množiny y, nesnižuje to využitelnost generátoru v úlohách typu Monte Carlo.

Generátorem normalizovaného rovnoměrného rozdělení tedy vyrobíme konkrétní hodnotu y a transformujeme ji na odpovídající x podle vzorce $x = F^{-1}(y)$.

Využití uvedené metody je omezeno na případy, kdy máme analyticky zadanou distribuční funkci $F(x)$ a navíc dokážeme jednoduše určit její inverzní funkci $F^{-1}(x)$. Jako příklad uvedeme *exponenciální rozdělení*, jehož distribuční funkce je $F(x) = 1 - e^{-\lambda x}$. Transformační vzorec zřejmě získáme symbolickým řešením rovnice

$$y = 1 - e^{-\lambda x}$$

vzhledem k neznámé x. Po logaritmování a úpravě dostaneme:

$$x = -\frac{1}{\lambda}\ln(1 - y)$$

Vzhledem k tomu, že soubor náhodných čísel vytvořených odečítáním prvků Y od jedničky má stejné (tj. normalizované rovnoměrné) rozdělení jako Y, používá se *výsledný transformační vzorec pro exponenciální rozdělení* v jednoduchém tvaru

$$x = -\frac{1}{\lambda}\ln(y), \tag{4.2}$$

kde y je konkrétní hodnota poskytnutá generátorem normalizovaného rovnoměrného rozdělení a x je konkrétní prvek souboru X čísel s exponenciálním rozdělením. Jako příklad k procvičení realizujte v jazyce **C** funkci generátoru náhodných čísel s exponenciálním rozdělením (prototyp `double negexp (double lambda)`).

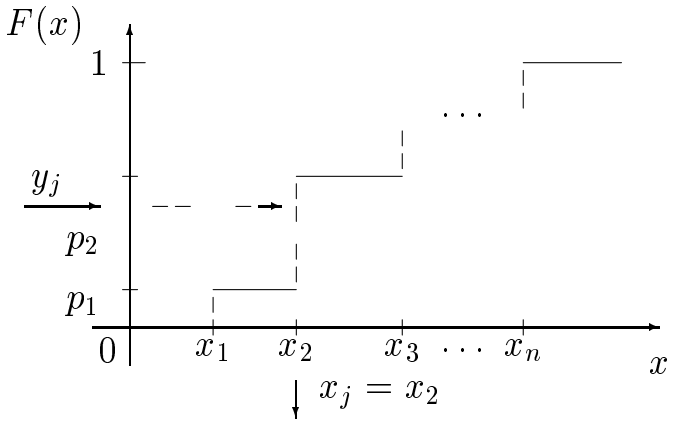
Dále ještě uvedeme transformační vzorec pro rovnoměrné rozdělení na obecném intervalu $\langle a, b)$. Metodu inverzní transformace lze v tomto případě použít intuitivně [9]. Číslo y poskytnuté základním generátorem je třeba „natáhnout“ na velikost intervalu $b - a$ a „odsadit“ od nuly o hodnotu a, tedy

$$x = a + (b - a)y \tag{4.3}$$

[9] Legální matematický postup zahrnující vytvoření analytického výrazu pro distribuční funkci rozdělení a symbolický výpočet inverzní funkce si proveďte z cvičných důvodů sami.

4.2.3 Diskrétní rozdělení

Princip metody inverzní transformace lze jednoduše využít i pro generování náhodných čísel X s diskrétním rozdělením. Tato čísla mohou v obecném případě nabývat n hodnot $\{x_i\}_{i=1}^n$, každá z těchto hodnot se v posloupnosti generovaných čísel objevuje náhodně s odpovídající pravděpodobností z množiny $\{p_i\}_{i=1}^n$, $\sum_{i=1}^n p_i = 1$. Realizace metody inverzní transformace pro případ diskrétního rozdělení je znázorněna na obrázku 4.3.



Obrázek 4.3: Princip generování čísel s diskrétním rozdělením

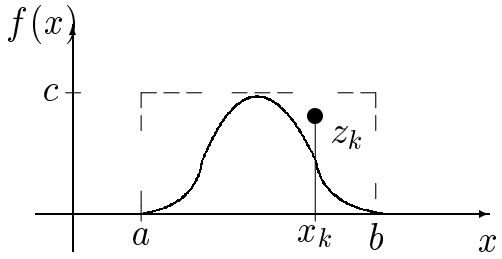
Tedy interval $\langle 0, 1)$ na ose reálných čísel rozdělíme na n dílčích intervalů s velikostí p_i. Základním generátorem vytvoříme konkrétní číslo z výběru Y (je v intervalu mezi 0 a 1) a zjistíme číslo i dílčího intervalu, ve kterém se vyskytuje. *Výstupem generátoru je odpovídající hodnota* x_i.

Jako příklad k procvičení realizujte v jazyce **C** funkci – univerzální generátor pro libovolné diskrétní rozdělení (nejprve si rozmyslete prototyp funkce).

4.2.4 Vylučovací metoda

Jedná se o experimentální metodu (tj. neodvozujeme žádný vzorec) vhodnou i pro případy, kdy není rozdělení náhodných čísel x zadáno analyticky. Pro popis rozdělení se využívá hustota pravděpodobnosti $f(x)$, která může být v konkrétním případě zadána pouze tabulkou hodnot. Předpokládáme, že

čísla x se vyskytují na intervalu $\langle a, b)$ a maximální hodnota $f(x)$ v tomto intervalu je c. K popisu metody využijeme obrázek 4.4.



Obrázek 4.4: Ilustrace vylučovací metody

Postup generování konkrétní hodnoty náhodného čísla s rozdělením $f(x)$ (tj. algoritmus generátoru založeného na vylučovací metodě) je následující:

- Generujeme náhodné číslo x_k z výběru s rovnoměrným rozdělením na intervalu $\langle a, b)$, tedy podle (4.3) $x_k = a + (b - a)y$, kde y je číslo poskytnuté základním generátorem.
- Generujeme náhodné číslo z_k z výběru s rovnoměrným rozdělením na intervalu $\langle 0, c)$, tedy podle (4.3) $z_k = cy$, kde y je (další) číslo poskytnuté základním generátorem.
- Čísla x_k, z_k interpretujeme jako souřadnice bodu v rovině (viz obr.4.4).
- *Pokud je náhodně generovaný bod pod křivkou funkce $f(x)$, postup končí a výstupem generátoru je hodnota x_k*, v opačném případě postup opakujeme počínaje prvním bodem. Index k v tomto případě slouží jako čítač obrátek cyklu opakování.

Uvedeným postupem zřejmě *vylučujeme* (též *odmítáme*) z výsledného výběru čísla s odpovídající malou hodnotou funkce hustoty pravděpodobnosti a preferujeme čísla s velkou hodnotou hustoty pravděpodobnosti. V podstatě se jedná o experimentální interpretaci (ve stylu Monte Carlo) smyslu funkce hustoty pravděpodobnosti.

4.2.5 Kompoziční metoda

Jako výchozí data pro kompoziční metodu opět slouží hustota pravděpodobnosti $f(x)$, *která nemusí být zadána analyticky*. Využití kompoziční metody

je výhodné v případech, kdy lze hustotu $f(x)$ rozumně aproximovat pomocí funkce po částech (tj. v dílčích intervalech základního intervalu výskytu čísel x) konstantní (popřípadě i lineární). Předpokládáme n dílčích intervalů, pravděpodobnost výskytu konkrétní hodnoty čísla x (tj. *plochu pod křivkou* $f(x)$) v i-tém dílčím intervalu označíme p_i.

Princip metody (tj. postup generování konkrétní hodnoty čísla x) vysvětlíme zhruba takto:

- Nejprve náhodně generujeme číslo dílčího intervalu i *pomocí generátoru s diskrétním rozdělením* (viz odst. 4.2.3) hodnot i podle pravděpodobností p_i.
- Dále generujeme náhodně číslo spadající do vybraného intervalu. Toto číslo generujeme podle charakteru funkce hustoty v příslušném intervalu. Je-li v nejjednodušším případě hustota pravděpodobnosti v intervalu konstantní (tj. rovnoměrné rozdělení), je (konečným) výstupem generátoru číslo s rovnoměrným rozdělením vypočítané podle vzorce $x = a_i + (b_i - a_i)y$, kde a_i a b_i jsou meze i-tého dílčího intervalu a y představuje konkrétní hodnotu poskytnutou generátorem normalizovaného rovnoměrného rozdělení.

Pro intervaly s lineárně rostoucí/klesající hustotou pravděpodobnosti lze s výhodu využít znalost z teorie pravděpodobnosti [10], a to že rozdělení maxima/minima ze dvou náhodně generovaných čísel z výběru y je lineárně klesající/rostoucí na intervalu $\langle 0, 1)$ (tj. grafické znázornění je trojúhelník). Jako příklad na procvičení se pokuste realizovat generátor pro pravděpodobnostní rozdělení $f(x)$ s charakterem lichoběžníku.

4.2.6 Generování čísel s normálním rozdělením

U některých rozdělení lze při generování čísel x využít jejich specifické vlastnosti. Konkrétně pro normální (gaussovské) rozdělení lze využít tzv. centrální limitní větu teorie pravděpodobnosti, která tvrdí, že součet náhodných čísel s libovolným rozdělením má asymptoticky [11] normální rozdělení se

[10] Uvádíme bez důkazů, pokuste se ověřit experimentálně ve stylu Monte Carlo. Dále se pokuste ověřit, že rozdělení maxima/minima ze tří čísel y má na intervalu $\langle 0, 1)$ charakter rostoucí/klesající paraboly, tj. mnohočlenu druhého stuně, atd. Ještě si rozmyslete, jak byste tuto skutečnost využili pro vylepšení kompoziční metody aproximací hustoty pravděpodobnosti po částech mnohočlenem druhého stupně.

[11] Blíží se s narůstajícím počtem prvků součtu.

střední hodnotou a rozptylem danými součtem středních hodnot a rozptylů pravděpodobnostních rozdělení prvků součtu [12].

Využijeme-li skutečnost, že pro normalizované rovnoměrné rozdělení je střední hodnota $E\{y\} = 0,5$ a rozptyl $D\{y\} = 1/12$, dostaneme součtem dvanácti konkrétních hodnot ze souboru y hodnotu x_s s (přibližně) normálním rozdělením se střední hodnotou 6 a rozptylem 1. Směrodatná odchylka (odmocnina z rozptylu) je též 1, rozdělení získané součtem můžeme symbolicky označit jako $N(6, 1)$. Pro konkrétní požadované normální rozdělení $N(a, \sigma)$ s parametry a (střední hodnota) a σ (směrodatná odchylka) lze pak nejprve provést transformaci rozdělení $N(6, 1)$ na *normalizované normální rozdělení* $N(0, 1)$ (jen posun) a poté transformaci na cílové rozdělení $N(a, \sigma)$ (posun a „roztažení"). Z těchto úvah rezultuje vzorec využitelný pro realizaci generátoru náhodných čísel s normálním rozdělením jako

$$x = a + \sigma((\sum_{j=1}^{12} y_j) - 6) \tag{4.4}$$

4.3 Zpracování výsledků

Při využití metod Monte Carlo náhodně generujeme parametry jednotlivých pokusů a vypočítáváme výsledky. *Číselné hodnoty výsledků je zřejmě opět nutno interpretovat jako konkrétní hodnoty (realizace) náhodných veličin.* Obvykle nás nezajímají rozsáhlé soubory dat – výsledků pokusů. Získanou informaci se snažíme redukovat do jednoduché podoby (několika málo) statistických charakteristik náhodných veličin figurujících jako výsledky.

Dále uvažujme jednu konkrétní náhodnou veličinu s charakterem výsledku, kterou označíme x. Provádíme N pokusů, výsledky pokusů tvoří množinu $\{x_i\}_{i=1}^{N}$. V dalších odstavcích se budeme zabývat postupy, pomocí kterých určíme z množiny výsledků pokusů odhady *střední hodnoty*, *rozptylu* a aproximaci *hustoty pravděpodobnosti* náhodné veličiny x.

4.3.1 Odhad střední hodnoty

V teorii pravděpodobnosti se odvozuje, že *nejlepším* odhadem střední hodnoty náhodné veličiny je *aritmetický průměr* z množiny realizací (hodnot

[12] Pro normální rozdělení nejde využít transformační metoda – není totiž možné jednoduše analyticky vyjádřit distribuční funkci rozdělení, natož pak funkci k ní inverzní.

získaných v jednotlivých pokusech), tedy

$$E\{x\} \doteq \bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{4.5}$$

Přímé využití tohoto vzorce předpokládá *ukládání výsledků pokusů x_i do jednorozměrného pole* a jednorázový výpočet průměru po dokončení všech pokusů. S ohledem na ušetření datové paměti potřebné pro výpočet (typicky sledujeme větší počet veličin–výsledků a provádíme řádově tisíce či desítky tisíc pokusů) preferujeme způsoby *průběžného výpočtu* statistických charakteristik sledovaných veličin.

V tomto případě zřejmě postačí zavést jedinou (tzv. *sumační*) proměnnou S_x, kterou před zahájením výpočtu nulujeme a v průběhu výpočtu k ní přičítáme výsledky jednotlivých pokusů. Dělení počtem pokusů N provedeme až na závěr výpočtu.

4.3.2 Odhad rozptylu

Rozptyl $D\{x\}$ je definován jako střední hodnota čtverce (tj. druhé mocniny) odchylky [13] realizací náhodné veličiny x od střední hodnoty $E\{x\}$. Odhad rozptylu lze (jednorázově, na závěr výpočtu) vypočítat z hodnot realizací jako

$$D\{x\} \doteq \frac{1}{N-1}\sum_{i=1}^{N}(\bar{x} - x_i)^2 \tag{4.6}$$

Pro velké hodnoty N lze pominout rozdíl mezi N a $N-1$, vzorec se zjednoduší a lze jej upravit do podoby vhodné pro průběžný výpočet:

$$\begin{aligned} D\{x\} &\doteq \overline{(x_i - \bar{x})^2} = \\ &= \frac{1}{N}\sum_{i=1}^{N}(\bar{x} - x_i)^2 = \frac{1}{N}\sum_{i=1}^{N} x_i^2 - 2\bar{x}\frac{1}{N}\sum_{i=1}^{N} x_i + \frac{1}{N}N\bar{x}^2 = \\ &= \Big(\frac{1}{N}\sum_{i=1}^{N} x_i^2\Big) - \bar{x}^2 \end{aligned} \tag{4.7}$$

[13] Směrodatná odchylka je definována jako odmocnina z rozptylu a jejím odhadem se zde tudíž nemusíme zabývat.

Pro průběžný výpočet rozptylu tedy potřebujeme zavést dvě sumační proměnné S_x a S_{x^2}, které před zahájením výpočtu nulujeme. V průběhu výpočtu realizujeme operace $S_x \leftarrow S_x + x_i$ a $S_{x^2} \leftarrow S_{x^2} + x_i^2$. Na závěr vypočítáme $\bar{x} = S_x/N$, $E\{x\} \doteq \bar{x}$ a $D\{x\} \doteq (S_{x^2}/N) - \bar{x}^2$.

Jistou chybou na kráse uvedeného postupu průběžného výpočtu rozptylu je snížení přesnosti vlivem zaokrouhlovacích chyb – do S_{x^2} se sčítá velký počet potenciálně velkých (čtverce) čísel [14]. Proto se využívají další metody průběžného výpočtu rozptylu, které tento nedostatek nemají. Jako příklad uvedeme užitečnou Hansonovu metodu [Oleh82], využívající jednu proměnnou M k průběžnému výpočtu aritmetického průměru a další proměnnou T pro průběžné sčítání čtverců (relativně malých) odchylek realizací od střední hodnoty (tj. od aktuálního průměru):

inicializace: $M_1 = 1, \quad T_1 = 0$

cykl pro $i = 1$ *až* N*:*

$$\begin{aligned} M_i &= \frac{i-1}{i} M_{i-1} + \frac{x_i}{i} \\ T_i &= T_{i-1} + \frac{i-1}{i}(M_{i-1} - x_i)^2 \end{aligned}$$

výsledky: $E\{x\} = M_N, \quad D\{x\} = \frac{T_N}{N-1}$

Protože téměř nic není zadarmo, je Hansonova metoda sice přesnější, ale za to výpočetně náročnější než předchozí uvedený postup – akce prováděné v každém kroku vyžadují větší počet tzv. dlouhých operací (násobení, dělení).

4.3.3 Konstrukce histogramu

Pokud chceme získat nějakou představu o charakteru pravděpodobnostního rozdělení náhodné veličiny–výsledku x, můžeme vytvořit tzv. *histogram* jako *po částech konstantní aproximaci* funkce hustoty pravděpodobnosti. Opět vycházíme z existující množiny výsledků N pokusů $\{x_i\}_{i=1}^N$ (vstupní data). Pro jednoduchost uvedeme postup předpokládající jednorázové zpracování uložených dat, modifikace pro průběžný výpočet není složitá a *opravdový programátor* ji lehce zvládne.

[14] Výpočet z uložených hodnot x_i podle (4.6) tímto nedostatkem netrpí, protože se sčítají (relativně malé) čtverce odchylek.

Hrubý postup rozčleníme do následujících bodů:

- Zjistíme základní interval $\langle a, b\rangle$, ve kterém se vyskytují prvky množiny $\{x_i\}_{i=1}^{N}$ [15].
- Základní interval rozdělíme na K (pokud možno stejně velkých) dílčích intervalů, velikost intervalu označíme Δ, intervaly číslujeme indexem například k. Počet dílčích intervalů bývá typicky řádu desítek. Nepochybně musí platit $K \ll N$ s ohledem na dostatečnou přesnost náhrady.
- Určíme počet c_k hodnot x_i vyskytujících se v každém intervalu k. Veličina c_k/N se nazývá *relativní četnost* výskytu x_i v k-tém intervalu a představuje *odhad pravděpodobnosti p_k, s kterou hodnota x v nějakém pokusu padne do příslušného intervalu.*
- Předpokládáme–li v k–tém intervalu konstantní (náhradní) hodnotu hustoty pravděpodobnosti f_k, z definice hustoty (plocha pod křivkou) vyplývá vztah $p_k = f_k\Delta \doteq c_k/N$. Z toho dostaneme jako výsledek odhad (střední hodnoty) hustoty pravděpodobnosti v k-tém intervalu jako $f_k = c_k/(N\Delta)$.

4.3.4 Statistické charakteristiky náhodných funkcí času

Složitější diskrétní simulační model nelze zpravidla redukovat na množinu pokusů a statistické vyhodnocení jejich výsledků. Obvykle se jedná o model dynamického systému vyvíjejícího se *v diskrétním čase*. Uvažujme například model založený na konceptu SHO (tj. nějakou síť front a obslužných uzlů). V tomto případě může být sledovanou a vyhodnocovanou náhodnou veličinou x například *doba průchodu požadavku sítí*. Pro tento případ jsou určeny předchozí odstavce - tj. výpočet odhadu střední hodnoty, rozptylu a histogramu z diskrétní množiny hodnot $\{x_i\}$ dob průchodu jednotlivých požadavků sítí. Jiná situace nastane, je-li sledovanou veličinou x například délka nějaké fronty, která se (náhodně) mění v modelovém čase. V tomto případě nemáme pro statistické vyhodnocení výsledků k dispozici diskrétní množinu hodnot, ale *náhodnou funkci modelového času* $x(t)$, kterou chápeme jako jednu z možných *realizací* náhodného procesu $X(t)$.

[15] Případně intuitivně „odfiltrujeme“ malý počet hodnot výrazně „vzdálených“ od ostatních.

Nebudeme se zde zabývat potřebnou teorií náhodných procesů (viz např. [Havr86], [Mand85]) a uvedeme pouze doporučení, jak provést potřebná statistická vyhodnocení. Omezíme se přitom na náhodné procesy, které jsou *stacionární* a *ergodické*. Základní charakteristikou náhodného procesu $X(t)$ je tzv. *první hustota pravděpodobnosti* $f(x,t)$, která udává hustotu pravděpodobnosti získanou z hodnot jednotlivých realizací $x(t)$ v časovém bodě t. Pro stacionární náhodný proces *nezávisí první hustota pravděpodobnosti na čase* a lze ji tedy popsat funkcí jedné proměnné $f(x)$. Ergodický náhodný proces musí být stacionární a navíc lze získat odhad první hustoty pravděpodobnosti $f(x)$ statistickým vyhodnocením *libovolné realizace* $x(t)$ *náhodného procesu* $X(t)$ [16].

Praktický postup statistického vyhodnocení náhodné funkce $x(t)$ spočívá v jejím náhodném *vzorkování*, tj. v získání množiny hodnot $\{x_i\}$, které funkce nabývá v náhodně zvolených časových bodech $\{t_i\}$. Obecně platí, že náhodný proces vzorkování (tj. způsob výběru hodnot t_i) nesmí být korelovaný se vzorkovaným procesem $X(t)$ (oba procesy musí být statisticky nezávislé) [17]. Hustotu vzorkování volíme tak, abychom pro vyhodnocení měli k dispozici řádově tisíce nebo desítky tisíc hodnot x_i [18]. Z těchto hodnot pak můžeme vypočítat odhad střední hodnoty, rozptylu a histogram dříve uvedenými postupy.

4.3.5 Problematika přesnosti výsledků

Prozatím jsme se spokojili s konstatováním, že k získání dostatečně přesných statistických charakteristik sledované náhodné veličiny je třeba mít k dispozici množinu výsledků $\{x_i\}_{i=1}^{n}$, kde číslo n je řádu tisíce nebo desítky tisíc. Toto tvrzení zde upřesníme pro střední hodnotu $E\{x\}$ náhodné veličiny x

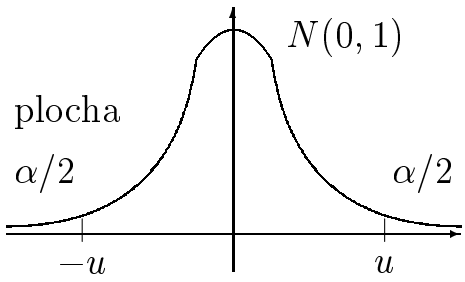
[16] Sledované náhodné funkce v simulačních modelech (viz např. zmíněná délka fronty) obvykle nesplňují předpoklad stacionárnosti, protože i v modelu určeném pro odhad stacionárních pravděpodobnostních charakteristik typicky zachycujeme také *přechodový děj* na počátku vývoje modelu (například startujeme model ze stavu s prázdnými frontami). *Modelujeme-li dostatečně dlouhou dobu „života systému“, lze vliv přechodného děje na výslednou statistiku zanedbat.*

[17] V některých případech může vyhovět i *ekvidistantní* (tj. pravidelné) vzorkování. Chceme-li si ušetřit starosti, volíme vzorkovací proces jako *poissonovský* (tj. perioda vzorkování je nezávislá náhodná veličina s exponenciálním rozdělením).

[18] V simulačním modelu programově koncipovaném na pseudoparalelních procesech (viz dále) je výhodné konstruovat pro účely vzorkování periodicky aktivovaný samostatný proces.

odhadovanou jako aritmetický průměr $\bar{x}$ z hodnot x_i. Takto odhadnutý průměr je opět náhodná veličina se střední hodnotou $E\{\bar{x}\} = E\{x\}$, rozptylem $D\{\bar{x}\} = D\{x\}/n = \sigma^2\{x\}/n$ a *normálním rozdělením pravděpodobnosti* [19]. Předpokládejme že chceme, aby absolutní chyba výsledku $\epsilon = |\bar{x} - E\{x\}|$ nepřekročila zvolenou hodnotu ϵ_{max} s pravděpodobností $1 - \alpha$ (tj. s pravděpodobností α ji překročí). Hodnota α představuje tzv. *hladinu významnosti*, typicky $\alpha = 0,05$ nebo $\alpha = 0,005$.

Pro zvolenou hodnotu hladiny významnosti α dokážeme určit tzv. *kritickou hodnotu* $u(\alpha)$ normalizovaného normálního rozdělení $N(0,1)$. Význam kritické hodnoty demonstruje následující obrázek.



α	u
$0,05$	$1,65$
$0,01$	$2,33$
$0,005$	$2,58$

Obrázek 4.5: Kritické hodnoty rozdělení $N(0,1)$

Chceme-li zaručit požadovanou přesnost odhadu střední hodnoty sledované náhodné veličiny x, postupujeme následovně:

- Kromě střední hodnoty $E\{x\}$ odhadujeme v průběhu simulačního experimentu také $\sigma\{x\}$ (standardním postupem – viz příslušné předchozí odstavce, směrodatná odchylka je odmocnina z rozptylu).

- Po vyhodnocení n hodnot x_i vypočítáme odhad směrodatné odchylky $\sigma\{x\}$ *a zanedbáme jeho chybu* (tj. rozdíl mezi odhadem a skutečnou hodnotou směrodatné odchylky). Dále se vypočítá *mezní chyba odhadu* odpovídající zvolené hladině významnosti α podle vzorce [20] :

[19] Využijeme poznatky z teorie pravděpodobnosti, týkající se střední hodnoty a rozptylu součtu náhodných veličin.

[20] Fakticky se jedná o transformační vztah, kterým „natahujeme“ kritické hodnoty normalizovaného rozdělení $N(0,1)$ (které známe) na odpovídající hodnoty *konkrétního normálního rozdělení* $N(0, \sigma\{x\})$, přičemž parametr tohoto rozdělení (směrodatnou odchylku) jsme experimentálně určili v průběhu modelování.

$$\epsilon = \frac{\sigma\{x\}}{\sqrt{n}} u(\alpha)$$

- Pokud je splněna nerovnost $\epsilon \leq \epsilon_{max}$, končíme proces modelování s tím, že *chyba odhadu střední hodnoty* $E\{x\}$ *je menší než zvolené* ϵ_{max}. *Přitom je pravděpodobnost* α, *že skutečná chyba odhadu je větší.*

- Pokud není nerovnost splněna, pokračujeme v procesu modelování (tj. zvětšíme rozměr n množiny $\{x_i\}$ a celý postup opakujeme).

Poznámka:

Existují postupy řízení výběru pokusů tak, aby se menším (relativně) počtem pokusů zvýšila přesnost odhadu střední hodnoty náhodné veličiny (výsledku pokusu). Tyto postupy jsou označovány jako *metody redukce rozptylu* a pro přesnější informace lze odkázat na literaturu [Laub87]. Při vytváření konkrétních simulačních programů se příliš neuplatňují, pravděpodobně proto, že vynaložené úsilí (a komplikace v programu) neodpovídá dosažitelnému zpřesnění odhadů střední hodnoty.

4.4 Objektově orientovaná dekompozice simulačního modelu

4.4.1 Diskrétní simulační model - konceptuální úroveň

Jak již bylo řečeno na počátku této kapitoly, diskrétní simulační modely reálných systémů (třeba model složitějšího dopravního či výpočetního systému na bázi SHO) nelze redukovat na triviální cykl metody Monte Carlo, tj. provedení N pokusů a jejich statistického vyhodnocení. První hrubý pohled na objektově orientovaný [21] diskrétní simulační model na konceptuální úrovni by mohl vypadat asi takto:

[21] První využití principů a prostředků pro objektově orientované programování bylo zaznamenáno již „dávno“ (vztaženo ke krátké historii počítačů a programování) právě v oblasti aplikací diskrétních simulačních modelů. Prvním (a filosoficky už v době vzniku dokonalým) prostředkem pro objektově orientované programování je SIMULA, původně SIMULA67 podle první úspěšné verze z roku 1967.

Model systému je tvořen množinou vzájemně vázaných objektů – *modelů prvků systému* a případně *modelů prvků okolí*. Základní vlastnosti prvků systému jsou modelovány jako *atributy* objektů, tj. jako vlastnosti s charakterem *dat*. Vazby objektů jsou realizovány prostřednictvím atributů s charakterem *odkazů*.

Stav modelu je dán aktuálními hodnotami všech nekonstantních atributů objektů (tj. je dán kompozicí stavů všech objektů) a mění se v diskrétních bodech *modelového* času vlivem *interakce objektů* (vzájemného působení). Interakce objektů je modelem interakce prvků systému a je realizována voláním *metod* (vlastností s charakterem procedur či funkcí) určitého objektu z jiného objektu.

Primárním důvodem interakce objektů je reakce na *událost*. Událost je objekt, jehož základní atributy jsou *čas vzniku události* a *typ události*. Události lze zhruba dělit na *vnější a vnitřní*. Vnější události generují objekty – modely okolí (např. příchody požadavků ze zdroje požadavků do obslužného systému s charakterem SHO). Vnitřní události vytváří objekty – modely prvků systému jako reakci na vnější událost nebo jinou vnitřní událost, obvykle s časovým posunem modelujícím *zpoždění* (dynamiku) prvků systému.

Z prezentovaného pohledu na diskrétní simulační model reálného systému není zřejmá dosud zdůrazňovaná souvislost s experimentálním pravděpodobnostním modelováním. Fakticky tato souvislost nemusí vůbec existovat - principiálně stejným (výše naznačeným) způsobem lze konstruovat i *deterministický* model, například *model logického systému* (objekty v modelu jsou hradla a klopné obvody, vazby odpovídají propojení hradel, vnější události jsou změny vstupních signálů, vnitřní události jsou změny signálů na vnitřních vodičích). Ve *stochastickém diskrétním modelu* jsou typicky časové body vzniku vnějších událostí náhodné, parametry událostí mohou být též náhodné a zpoždění v sekvencích (*příčina–důsledek*) vnitřních událostí jsou rovněž náhodná. V důsledku toho se *veličiny popisující stav modelu mění v náhodných časových bodech (tj. v diskrétním čase) a mají tedy charakter náhodných funkcí času* (*náhodných procesů*). Pro stochastické diskrétní simulační modely se tedy využívá numerické řešení ve stylu Monte Carlo v tom smyslu, že (náhodně se chovající) model se stimuluje dlouhou dobu (v modelovém čase) náhodnými podněty a vybrané veličiny charakterizující jeho chování se statisticky vyhodnotí.

4.4.2 Řídicí algoritmus simulačního výpočtu

Prezentovaný pohled na diskrétní simulační model neřeší problém řídicího algoritmu simulace, tj. algoritmu, který modelem „hýbá“. Existují dva základní přístupy, obvykle označované jako *metoda interpretace událostí* a *metoda (pseudo)paralelních procesů*.

Metodu **interpretace událostí** lze (nad dříve charakterizovaným modelem) ve stručnosti popsat zhruba takto:

> Všechny události, *které mají nastat v budoucím vývoji modelu* (vztaženo k aktuální hodnotě modelového času) jsou vedeny v seznamu nazývaném *kalendář událostí*. Seznam je setříděn vzestupně podle hodnoty modelového času vzniku události (atribut události). Řídicí algoritmus simulačního výpočtu (realizovaný například v podobě hlavního programu) postupně *interpretuje* události nacházející se v kalendáři. Interpretace probíhá v diskrétním bodě modelového času vzniku události a je realizována vyvoláním *interpretačního podprogramu* příslušného k typu události. Interpretační podprogram provede změny v modelu (tj. změny hodnot atributů objektů) voláním metod objektů, kterých se událost týká. Pokud je interpretovaná událost přímou *příčinou* další události v budoucím modelovém čase, je součástí činnosti některé volané metody objektu *naplánování* (tj. zatřídění do kalendáře) této „nové“ události. Interpretovaná událost je vymazána z kalendáře.

Pro realizaci tohoto způsobu řízení výpočtu [22] vystačíme s filosofií objektů poskytovanou konvenčními prostředky pro objektově orientované programování (třeba **C++**, **Eiffel**, **Smalltalk**):
Objekt je množina atributů (tj. prvků s charakterem dat), jejichž konkrétním obsahem je určen stav objektu. S objektem je sdružena množina metod (operací), které umožňují vnější přístup k atributům. Objekt je sám o sobě pasivní (nic nedělá), ale na požádání (volání jeho metody) poskytuje pro okolí určité služby. [23]

[22] Fakticky nelze dekompozici modelu (a odpovídající způsob řízení výpočtu) založený na metodě interpretace událostí považovat za „čistě objektový“. Podprogramy pro interpretaci událostí *nelze v obecném případě jednoznačně přiřadit jako metodu k žádnému z objektů modelu*, každý z podprogramů obecně „operuje“ nad celým modelem.

[23] V případě potřeby ale můžeme samostatnou činnost objektu realizovat v rámci konstruktoru nebo vybrané metody a předávání řízení mezi objekty realizovat způsobem popsaným dále v odst. 5.1.4.

Metoda **paralelních procesů** je založena na objektové dekompozici řídicího algoritmu simulačního výpočtu. Jednotlivé části výpočtu jsou zapouzdřeny ve vybraných typech (tj. *třídách*) objektů, které tím získávají (navíc k atributům a metodám) vlastní program a charakter samostatných výpočetních procesů vykonávaných *pseudoparalelně* (tj. jejich exekuce je „prostřídaná" v diskrétním modelovém čase). Opět se pokusíme stručně vystihnout filosofickou podstatu metody paralelních procesů (nad dříve charakterizovaným simulačním modelem diskrétního systému):

Některé objekty modelu mají *pasivní* charakter, tj. nevyvíjí v modelovém čase žádnou vlastní aktivitu a pouze poskytují služby (realizované voláním jejich metod) pro jiné objekty. Jiné modely objektu, dále označované jako *procesy*, mají aktivní charakter, tedy vyvíjí (souběžně s ostatními procesy) aktivitu *podle vlastního programu*.

Aktivita procesu je členěna do sekvence tzv. *fází aktivity*, probíhajících vždy v jednom konkrétním bodě diskrétního modelového času. Výsledkem činnosti – fáze aktivity je změna atributů (lokálních dat) objektu – procesu, případná změna atributů jiných objektů a případná změna stavu jiných procesů (např. jejich aktivace). Intervaly modelového času mezi fázemi aktivity určitého procesu se nazývají *úseky nečinnosti*.

Řízení pseudoparalelního výpočtu všech procesů v modelu opět vyžaduje existenci datové struktury s charakterem kalendáře. V kalendáři jsou vzestupně podle hodnoty modelového času setříděny události, jejichž záznam obsahuje (kromě hodnoty času) ještě odkaz do programu procesu (tzv. *reaktivační bod*). Řídicí smyčka výpočtu postupně spouští procesy v pořadí daném záznamy v kalendáři [a].

[a] Větší detaily viz *korutiny* v popisu SIMULY (podkap. 4.6) a v popisu implementace C-Sim v kap 5.

Hlavní výhodou dekompozice modelu založené na aktivních objektech – procesech je větší schopnost modelovat realitu. Objekty reálného světa jsou totiž často aktivní – vykonávají vlastní činnost, která probíhá souběžně (paralelně) s činností jiných objektů a modelovat je programovými objekty s charakterem výpočetních procesů je přímočaré a přirozené. Navíc *dekompozice je čistě objektová*, protože program vykonávaný objektem – procesem lze jednoznačně přiřadit (*zapouzdřit*) k jeho třídě. Čistě objektová

dekompozice přináší všechny výhody objektového přístupu v programování - zejména pak výhodu *opakovatelné využitelnosti* (angl. *reusability*) programového kódu. Na druhé straně ale *vyžaduje využití metody paralelních procesů obecnější model objektu* než poskytují jazyky s charakterem **C++**. Konkrétně se jedná o *doplnění programu procesu do deklarace třídy objektu – procesu.*

Toto bylo poprvé realizováno v programovacím jazyce *SIMULA* (původně **SIMULA** 67) [Dahl72], [Stau78]. Jazyk má velmi čistou a propracovanou koncepci a dosud se poměrně často využívá. Jako hlavní charakteristiky lze uvést: odvozen od **Algolu 60**, využívá se jen jednoduchá dědičnost, objekty se vytváří pouze dynamicky a odkazuje se na ně výhradně pomocí referenčních proměnných, uvolňování dynamické paměti nepoužívaných objektů provádí automaticky *čistič paměti* (angl. *garbage collector*). Nevýhodou **SIMULY** je příbuzenský vztah k **Algolu 60**, který již byl vývojem překonán a neužívá se [24]. Proto je také dále v kap. 5 prezentován programovací nástroj **C-Sim** využívající filosofické základy **SIMULY**, ale koncipovaný jako rozšiřující (externí) knihovna k jazyku **C**.

4.4.3 Základní třídy objektů pro modelování SHO

Procesově orientovaná diskrétní simulace je v podstatě univerzální číslicovou modelovací technikou, s jejíž pomocí lze realizovat modely diskrétní i spojité [25], deterministické i stochastické a jejich nejrůznější kombinace. V dalším textu se budeme orientovat zejména na problematiku modelování reálných systémů s charakterem SHO, tedy na *stochastické modely* vyvíjející se v *diskrétním modelovém čase.*

V simulačním modelu založeném na metodě paralelních procesů v první řadě rozlišujeme objekty *pasivní* (nemají vlastní program, realizují zapouzdření atributů (tj.dat) a metod (tj. operací či služeb využitelných z vnějšku objektu) a objekty *aktivní*, které navíc mají vlastní program vykonávaný pseudoparalelně s programy jiných takových objektů. Další rozlišení objektů v simulačním modelu může být na *statické* (existují po celou dobu výpočtu

[24] Současným navázáním na skandinávskou tradici jazyků Algol60, SIMULA67, SIMULA a Algol68 je BETA. Tento objektově orientovaný jazyk má jednotnou abstrakci (označenou jako *pattern*, tj. *vzor*) pro procedury, procesy, objekty, třídy a výjimky.

[25] Což je na první pohled překvapivé, ale jen do té doby, než si uvědomíme, že diferenciální rovnice představující spojitý matematický model reálného systému se řeší metodami numerické integrace v *diskrétním* čase.

simulačního programu) a *dynamické*, které jsou vytvářeny a rušeny (jinými objekty s charakterem procesů) průběžně. V první fázi vytváření simulačního modelu se musíme rozhodovat, jaké typy (tj. třídy objektů) použijeme, kterým dáme aktivní či pasivní charakter a zda nemůžeme svoje potřebné typy odvodit (děděním) z nějakých již existujících typů. Speciálně objektům využívaným pro modelování SHO často potřebujeme dát vlastnosti *prvku seznamu* (například objekty modelující *požadavky* se musí řadit do front realizovaných jako seznamy). Dále zřejmě potřebujeme objekty s charakterem *seznamů*, které využijeme jako objekty modelující fronty.

Příkladem budiž opět programovací jazyk **SIMULA**. V **SIMULE** mohou být objekty vytvářeny jen v dynamicky přidělované paměti a reprezentovány jen prostřednictvím tzv. *referenčních proměnných* (analogie ukazatelů v **C**). K základnímu jazyku mohou být připojeny tzv. *systémové třídy* `SIMSET` a `SIMULATION`. V `SIMSET` jsou soustředěny prostředky umožňující práci s *obousměrnými cyklickými seznamy*. Jedná se o třídy `LINKAGE` (nepoužívá se přímo, sdružuje společné vlastnosti tzv. *hlavy* seznamu a *prvků seznamu*), `LINK` (potomek `LINKAGE`, doplňuje specifické vlastnosti (jen metody) prvku seznamu) a `HEAD` (opět potomek `LINKAGE`, hlava (tj. reprezentant) seznamu, doplňuje specifické vlastnosti – operace nad celým seznamem). *Při konstrukci simulačního modelu se často fronty přímo modelují objektem se základním typem* `HEAD`. *Naproti tomu je vždy třeba vytvořit vlastní typy (např. objektů modelujících požadavky) odvozením od* `LINK` *a přidat vlastní atributy (tj. informační obsah prvku seznamu).*

Pro přehled uvedeme významné abstraktní vlastnosti s charakterem metod pro třídu `LINK` [26]:

into (seznam) – voláním metody (například *prvek.into (sez1)*) řadíme objekt – prvek seznamu na *konec* seznamu – parametru metody.

follow (jiný_prvek) – voláním metody se řadí aktuální objekt do seznamu *za* objekt *jiný_prvek* [27].

precede (jiný_prvek) – voláním metody se řadí aktuální objekt do seznamu *před* objekt *jiný_prvek*.

[26] Jejich detailněji popsané ekvivalenty pro **C–Sim** viz kap. 5.

[27] Seznam není třeba specifikovat, operace se provede jen nad spojkami (*dopředu, dozadu*) aktuálního objektu a objektu *jiný_prvek*.

out() – volání metody vyřadí aktuální prvek ze seznamu (k provedení operace opět není třeba specifikovat konkrétní seznam).

Ještě uvedeme významné abstraktní vlastnosti (s charakterem operací) pro třídu `HEAD`. Jedná se o operace realizovatelné nad celým seznamem:

empty() – test na prázdný seznam, vrací booleovskou hodnotu *true*, je–li seznam prázdný, jinak vrací *false*.

cardinal() – metoda vrací délku (tj. počet prvků) seznamu.

first() – metoda vrací odkaz na první prvek seznamu.

last() – metoda vrací odkaz na poslední prvek seznamu.

clear() – volání metody vyprázdní seznam [28].

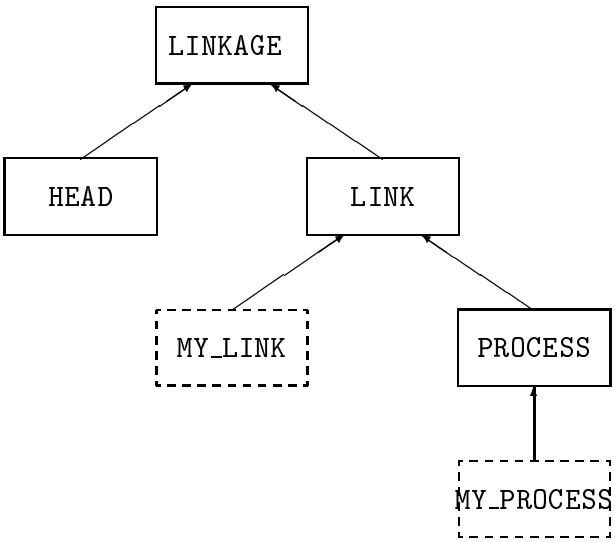
V systémové třídě **SIMULATION** je deklarována třída `PROCESS` jako potomek `LINK`, tj. procesy jdou řadit do seznamů – front. *Při vytváření simulačního modelu je třeba odvodit od* `PROCESS` *vlastní třídy aktivních objektů – tj. dodat atributy (lokální data procesu), případně přidat metody, ale určitě přidat program chování procesu.* Vlastnosti a stavy procesů realizovaných ve třídě PROCESS upřesníme v další podkapitole. Konkrétní vlastnosti procesů dostupné při použití **C–Sim** jsou popsány v kap. 5.

Ještě v obrázku 4.6 uvedeme schematicky příbuzenské vztahy základních tříd využitelných pro konstrukci simulačního modelu sítě front v **SIMULE**. Čárkovaně jsou naznačeny typy, které se při konstrukci simulačního modelu musí doplnit.

4.5 Procesy v programovacím jazyce SIMULA

Významný rozdíl **SIMULY** proti **C++** a jiným používaným objektově orientovaným jazykům spočívá v doplnění operační části (tj. posloupnosti příkazů) do deklarace třídy. Syntakticky má deklarace například třídy `AB` následující podobu:

[28] V **SIMULE** (na rozdíl příkladně od kontejnerové knihovny Classlib pro **C++**) není třeba se o „osud“ prvků vyprázdněného (či zrušeného) seznamu nijak starat, je to záležitost *čističe paměti*, který uvolní dynamickou paměť přidělenou pro objekt, *na který neexistuje žádný platný odkaz.*



Obrázek 4.6: Základní typy objektů pro konstrukci modelu sítě front

```
A class AB (formální_parametry)
    specifikace_typu_parametrů;
  begin
    deklarace_atributů;
    deklarace_metod;
    operační_část_AB;
  end;
```

V uvedené deklaraci představuje `A` jméno bázové třídy. Parametry odpovídají parametrům konstruktoru v **C++** s tím rozdílem, že je lze využívat jako řádné atributy po celou dobu existence objektu. Deklarace atributů a metod (mají formu procedur obvyklou v **Algolu 60**) není nijak filosoficky odlišná od **C++**. Hlavní rozdíl proti obvyklému chápání objektů přináší *operační_část_AB*, která algoritmicky popisuje specifickou část „života“ objektu třídy `AB`. Objekty této třídy dědí operační část svého předka (tj. `A`), jejíž vykonání předchází vykonání vlastní operační části. V této souvislosti je vhodné uvést, že využití vícenásobné dědičnosti by přineslo problém stanovení pořadí, ve kterém budou vykonány zděděné operační části předků. Je to jeden z důvodů, proč jazyky následující po **SIMULE** (např. **C++**, **Eiffel**) využily jednodušší model „pasivních“ objektů.

Operační části všech existujících objektů by měly být vykonávány paralelně (na jednoprocesorovém počítači *pseudo–paralelně*). Pro implementaci

pseudo–paralelního výpočtu byl v **SIMULE** využit koncept *korutin* (z angl. *coroutines*). Před vysvětlením tohoto stylu programování připomeneme klasický procedurální styl (**Algol**, **Fortran**, **Pascal**, **C**) využívající *subroutiny* (procedury, funkce) volané z hlavního programu nebo vzájemně. Vztah např. hlavního programu a volané procedury je asymetrický – volaná procedura neví kdo ji vyvolal, proběhne celá až do konce a vrátí řízení do místa vyvolání (anonymně operací `return`). Naproti tomu v programu koncipovaném jako množina korutin (tj. spolupracujících procedur) jsou všechny korutiny „rovnoprávné“ a jejich výpočet neprobíhá jednorázově, ale po částech. Po ukončení určitého úseku svého programu předává korutina `x` řízení jiné (explicitně pojmenované) korutině `y` operací např. `resume(y)` . Až získá `x` znovu řízení, pokračuje výpočet nikoliv od počátku, ale od příkazu následujícího za `resume(y)` (tzv. *reaktivační bod*). Předávání řízení mezi korutinami je „dobrovolné“ (tj. délka úseku výpočtu není nijak omezena).

V této fázi úvah se dostáváme k nutnosti rozlišovat korutinu jako programový text a výpočetní proces podle tohoto textu probíhající. Podle jednoho programu (musí být konstruován jako *reentrantní*) totiž může současně probíhat několik procesů. Operační část třídy `A` v **SIMULE** představuje programový text korutiny `k_A`. Výpočet operační části objektu (například `a1`) představuje výpočet procesu (označíme `p_a1`) podle programu `k_A`. Všechny objekty třídy `A` tedy „sdílí“ programový text korutiny `k_A`. Podle tohoto programového textu současně probíhají všechny existující procesy `p_ai`, každý ale pracuje nad unikátními atributy objektu `ai`), popřípadě sdílí globální data definovaná mimo třídu `A`.

Program v **SIMULE** lze pak chápat jako množinu kooperujících pseudo–paralelních výpočetních procesů. Na počátku výpočtu existuje pouze jeden proces (má vyhražené jméno `main`) probíhající podle hlavního programu. Uvedeme nyní několik málo podrobností nutných k lepšímu pochopení celé záležitosti.

Objekt je vytvořen (např. v procesu hlavního programu) dynamicky následující operací:

`a :-- new A (`*aktuální_parametry*`);`

Zde `a` je referenční proměnná deklarovaná jako `ref(A) a` , tj. je oprávněná (tzv. *kvalifikovaná*) odkazovat na objekty třídy `A a` jejích podtříd (polymorfismus). Pro přiřazení referencí se využívá speciální operátor `:--` Operátorem `new` se podobně jako v **C++** dynamicky vytváří nový objekt třídy `A`,

kterému se předají aktuální parametry. V C++ dojde k vyvolání konstruktoru, zde je předáno řízení na začátek korutiny `k_A`. Pokud je uvnitř korutiny pouze provedena inicializace atributů příslušného objektu, je výsledný efekt stejný jako při vyvolání konstruktoru v C++. Výpočet procesu `p_a1` doběhl do konce, nelze v něm již dále pokračovat, data objektu pochopitelně zůstávají v platnosti a lze je využívat voláním metod třídy `A` pro objekt `a1`. Zajímavější je situace, kdy jedním z příkazů použitých v programovém textu `k_A` je `detach` (tj. odpoj). Dojde k takzvanému „osamostatnění" objektu a jeho výpočetní proces se stává složkou množiny pseudo–paralelních procesů realizujících celkový výpočet. Samostatnému objektu (přesněji odpovídajícímu výpočetnímu procesu) lze předat řízení voláním `resume(a1)` z jiného samostatného objektu.

Operace pro řízení pseudo–paralelního výpočtu na úrovni základního jazyka (`detach, resume`) byly dále využity pro vytvoření dokonalejších prostředků formou tzv. systémových tříd. Ve třídě `SIMSET` jsou k dispozici třídy pro práci s cyklickými obousměrně zřetězenými seznamy (`LINKAGE` – sdružuje společné vlastnosti prvků a hlavy seznamu, `LINK` – prvky seznamu, `HEAD` – hlava seznamu). Ve třídě `SIMULATION` byly zavedeny výpočetní procesy jako instance třídy `PROCESS` (odvozené od `LINK` – proces tedy může být prvkem seznamu). Hlavní program je rovněž proces s vyhrazeným jménem `main`.

Procesy realizují svoji činnost v modelovém čase, jehož hodnotu je možné kdykoliv zjistit voláním operace `time`. „Život" procesů v modelovém čase má diskrétní charakter – *fáze aktivity* procesu se realizují v diskrétní hodnotě modelového času (mají nulovou délku trvání) a mezi nimi jsou *úseky nečinnosti* s nenulovou dobou trvání v modelovém čase. Základní datovou strukturou využívanou pro řízení pseudo–paralelního výpočtu je tzv. *kalendář*. V kalendáři jsou zřetězeny (v pořadí stoupajícího modelového času) záznamy obsahující konkrétní hodnotu modelového času a odkaz na proces, který má v tomto čase převzít řízení. Aktivní je vždy proces nacházející se v čele kalendáře.

Vytvořené procesy se mohou nacházet v následujících stavech:

Pasivní – není určen čas provedení příští fáze aktivity (proces nemá záznam v kalendáři).

Plánovaný – je určen čas provedení příští fáze aktivity (proces má záznam v kalendáři).

Aktivní – právě se provádí některá fáze aktivity (proces má záznam v čele kalendáře).

Ukončený – už byly provedeny všechny fáze aktivity (proces nejde znovu spustit, data zůstávají v platnosti).

K přechodům mezi stavy dochází provedením některé z následujících operací v aktivním procesu (zjednodušeně):

passivate – aktivní proces přechází do stavu **pasivní** (je vyřazen z kalendáře a je aktivován proces z čela kalendáře událostí).

hold – aktivní proces přechází do stavu **plánovaný** (a je aktivován následující proces podle kalendáře událostí). Délka následujícího úseku nečinnosti je parametrem operace.

activate – aktivní proces `x` převádí jiný proces `y` ze stavu **pasivní** do **plánovaný**. Aktivní proces `x` dále pokračuje v činnosti. Parametry operace jsou odkaz na `y` a časový údaj.

cancel – aktivní proces x převádí jiný proces `y` ze stavu **plánovaný** do **pasivní**. Aktivní proces dále pokračuje v činnosti. Parametrem operace je odkaz na `y`.

Kapitola 5

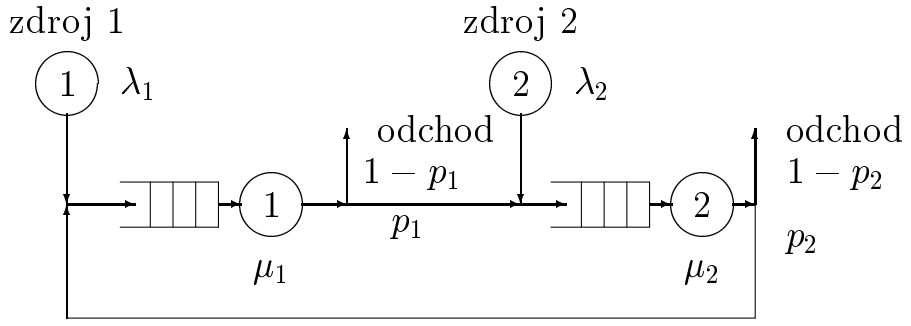
Příklady použití C-Sim

Poznámka:

C-Sim je simulační nástroj založený na jazyce **C** (ANSII norma). Poskytuje podobně jako **SIMULA** základní typy `LINK` - prvky seznamů, `HEAD` - seznamy, `PROCESS` - pseudoparalelní procesy (deterministická vlákna, která ve svém programu sama určují, kdy řídicí algoritmus přepne na jiné vlákno). Veškeré informace o **C-Sim** (včetně anglicky psané referenční příručky a možnosti natažení zdrojového kódu) jsou dostupné na `http://www.c-sim.zcu.cz`. Dále jsou uvedeny pouze komentáře k základním příkladům dostupným na sdíleném adresáři k předmětu VSP.

5.1 Simulační model otevřené sítě front

Uvažujme otevřenou síť front znázorněnou na následujícím obrázku.



Obrázek 5.1: Příklad sítě front

Síť obsahuje dva obslužné systémy, má dva vstupy požadavků z okolí (tj. dva vstupní proudy) a požadavky mohou opustit síť ve dvou místech (dva výstupní proudy). Předpokládejme, že všechna využitá pravděpodobnostní

rozložení jsou exponenciální s parametry λ_1, λ_2 (střední frekvence vstupních proudů) a μ_1, μ_2 (podmíněné střední frekvence obsluh). Tyto veličiny jsou zároveň vstupními parametry modelu. Jako další vstupní parametry budou využity pravděpodobnosti p_1 a p_2, tj. pravděpodobnosti, s kterými požadavek odchází ze systému po ukončení obsluhy v prvním resp. druhém obslužném systému. Simulační model bude konstruován s cílem určit *střední dobu odezvy* T_q, tj. střední dobu průchodu požadavku sítí.

Při vytváření modelu využijeme principy objektově orientované analýzy. Budeme se snažit nalézt *třídy* (tj. typy) prvků modelu a popsat je tak, aby byly univerzálněji využitelné (idea *reusability*), například i pro model podobné sítě složené ze tří či více obslužných uzlů. Rozlišíme následující typy prvků modelu:

Požadavek

Rozhodneme se pro pasivní model požadavku (tj. modelujeme jej pouze jako data). Vzhledem k tomu, že požadavky řadíme do front, odvodíme jejich typ od standardního typu `LINK`. Jako dodatečný atribut (navíc proti `LINK`) přidáme (s ohledem na cíl sledovaný konstrukcí modelu) *čas vstupu požadavku do SHO*. Typ *požadavek* vyjádřený v syntaxi **C-Sim**u tedy je:

```
typedef struct {
  d_link;
  double taktiv;      /* čas vzniku požadavku */
} TRNS_DATA;
```

Fronta požadavků

Frontu opět modelujeme jako pasivní (tj. nevyvíjí žádnou vlastní aktivitu) objekt odvozený od standardního typu `HEAD`. Navíc ke standardním vlastnostem tohoto typu (implementuje obousměrný cyklický seznam) přidáme ještě *vazbu (odkaz) na navazující prvek* – kanál obsluhy. Typ *fronta* vyjádřený v syntaxi **C-Sim**u bude:

```
/* datovy zaznam fronty - pasivni objekt */
typedef struct {
  d_head;
```

```
  PROCESS *obsluha;  /* ukazatel na proces obsluhy*/
} MY_HEAD;
```

Zdroj požadavků

Zdroj požadavků budeme modelovat jako aktivní objekt, tj. jako *proces* generující poissonovský proud požadavků. Odvodíme jej od standardního typu PROCESS přidáním atributů *střední frekvence proudu* a *vazba na navazující prvek* – frontu. Lokální data procesu popíšeme následujícím typem:

```
typedef struct {
  d_process;
  float lambda;       /* intenzita příchodů */
  MY_HEAD *fronta;    /* navazující fronta */
} PRICHOD_DATA;
```

Objekt s charakterem procesu nestačí popsat pouze prostřednictvím jeho *atributů* (tj. vlastností s charakterem lokálních dat), potřebujeme popsat také jeho činnost. Reentrantní (tj. může podle něj probíhat více procesů) program činnosti procesu má v **C–Sim** následující formu:

```
s_program(PRICHOD_DATA, PRICHOD_PROG)
static TRNS_DATA *p_poin; /* pomocný ukazatel */
  for (;;) {
    p_poin = s_new_link (TRNS_DATA);
    p_poin->taktiv = TIME();
    into ((LINK *) p_poin, (HEAD *) my.fronta );
    if (idle (my.fronta->obsluha))
      activate_at (my.fronta->obsluha, TIME());
    hold(negexp(my.lambda));
  }
s_end_program
```

Kanál obsluhy.

Jako poslední (a nejsložitější) typ prvků modelu zavedeme kanál obsluhy. Budeme jej modelovat jako objekt s vlastní činností, tj. odvodíme

jej od standardního typu PROCESS. Přidáme atributy s charakterem *střední frekvence obsluh*, *pravděpodobnosti odchodu ze systému po ukončení obsluhy*, *vazby–odkazu na předchozí prvek* a *vazby–odkazu na navazující prvek*. Dále přidáme atributy potřebné pro realizaci statistického sledování chování prvku, konkrétně *čítač požadavků* prošlých obsluhou a *součet dob průchodů požadavků* opouštějících systém [1]. Lokální data procesu popíšeme následujícím typem:

```
typedef struct {
  d_process;
  float mi;            /* intenzita obsluhy */
  float p;             /* pravděp. odchodu ze syst. */
  MY_HEAD *in_fronta;  /* vstupní fronta */
  MY_HEAD *out_fronta; /* navazující fronta */
  UWORD cit;           /* čítač průchodů */
  double stq;          /* součet dob průchodů */
} OBSLUHA_DATA;
```

Reentrantní program činnosti procesu má v **C–Sim** následující formu:

```
s_program(OBSLUHA_DATA, OBSLUHA_PROG)
static TRNS_DATA *p_poin; /* pomocný ukazatel */
  for (;;) {
    if (!empty((HEAD *) my.in_fronta)) {
      hold(negexp(my.mi));
    }
    else {
      passivate();
      continue;
    }
    p_poin = (TRNS_DATA *) first((HEAD *) my.in_fronta);
    if (((float)rand()/MAX_INT) > my.p) {
      my.cit ++;
      my.stq += TIME() - p_poin->taktiv;
      dispose_link((LINK*) p_poin);   /* zároveň out() */
    }
```

[1] Při konstrukci modelů prvků se zamýšleným univerzálnějším využitím je vhodné distribuovat data pro statistiku do objektů–procesů a jejich aktualizaci do programů procesů (zásada *zapouzdření*).

```
    else {
      out((LINK *) p_poin);
      into((LINK *) p_poin, (HEAD *) my.out_fronta);
      if (idle (my.out_fronta->obsluha))
        activate_at (my.out_fronta->obsluha, TIME());
    }
  }
s_end_program
```

V inicializační části simulačního programu realizujícího model konkrétní sítě generujeme příslušné objekty a provedeme jejich propojení. Konkrétněji se jedná o následující činnosti:

- Čtení dat – parametrů modelu.
- Vytvoření objektů (v **C-Sim** jen dynamicky) a naplnění ukazatelů (globálních dat simulačního programu) odkazy na vytvořené objekty.
- Inicializace dat – atributů objektů. V **C-Sim** jsou atributy vždy přístupné zvnějšku objektu (jako *public* v **C++**) a lze je naplnit přímým přiřazením. Je pochopitelně možné konstruovat pro každý použitý typ objektu inicializační funkci (analogie konstruktoru).
- Propojení prvků modelu se realizuje dosazením hodnot globálních ukazatelů (reprezentace stabilních objektů) do lokálních atributů objektů s charakterem odkazů.
- Provede se počáteční aktivace prvků s charakterem procesů. Alespoň jeden proces musí být na začátku aktivován. V popisovaném modelu sítě front musí být na začátku aktivovány procesy modelující zdroje požadavků.

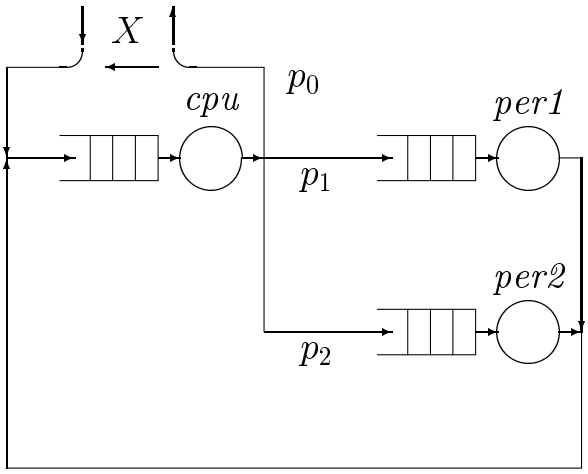
Kompletní programový text modelu (soubor OPENQN.C) je k dispozici ve sdíleném adresáři určeném pro informace o cvičeních z VSP. Příklad může po prostudování sloužit jako výchozí jednoduchý text pro vlastní experimenty. Speciálně lze doporučit:

- Vyzkoušet (změny parametrů) problematiku zatížení sítě (*stacionární režim* versus *zahlcení*).

- Vyzkoušet využití jiných než exponenciálních pravděpodobnostních rozdělení.
- Sestavit síť s jiným počtem a propojením prvků.
- Doplnit do modelu další typy prvků, například kanály obsluhy, z kterých požadavek může odejít do dvou různých (pravděpodobnost větvení) navazujících obslužných uzlů.

5.2 Model multiprogramního výpočetního systému

Komentovaný zdrojový text modelu je obsažen v souboru `CESEM.C`, který je k dispozici ve sdíleném adresáři určeném pro cvičení z předmětu VSP. Jedná se o simulační model výpočetního systému pracujícího v multiprogramním režimu činnosti (zkratka *CEntral SErver Model*). Zdrojový text modelu předpokládá využití interaktivního prostředí pro práci s **C-Sim** na počítači typu PC. Model je schematicky znázorněn na následujícím obrázku.



Obrázek 5.2: Model multiprogramního režimu

Výpočetní systém se skládá z procesoru (`cpu`) a dvou periferních zařízení (`per1`, `per2`). Všechny uvedené moduly jsou modelovány jako elementární systémy hromadné obsluhy s frontou typu FIFO bez priorit. Pravděpodobnostní rozložení doby obsluhy je exponenciální pro všechny subsystémy (parametry `micpu`, `miper1`, `miper2`).

Systém zpracovává pevný počet zakázek (pevný stupeň multiprogramování, počáteční nastavení na 1). Zakázky jsou dvou různých typů, které se liší hodnotami pravděpodobností větvení po opuštění obsluhy v cpu. Tyto pravděpodobnosti jsou (pro zakázku *i*-tého typu, *i=1,2*) označeny v programu:

p0*i*	pravděpodobnost ukončení zakázky po opuštění cpu
p1*i*	pravděpodobnost přechodu zakázky z cpu do obsluhy v per1
p2*i*	pravděpodobnost přechodu zakázky z cpu do obsluhy v per2

Pravděpodobnost výskytu zakázky prvního typu v multiprogramním mixu je q, druhý typ se vyskytuje s komplementární pravděpodobností 1-q. Po ukončení obsluhy v periferním zařízení se zakázky vrací opět do obsluh v cpu. Zakázky jsou modelovány pomocí objektů typu JREC definovaných jako rozšíření LINK o atributy p0*i*, p1*i*, p2*i* (pravděpodobnosti větvení pro příslušný typ zakázky).

Obslužný subsystém cpu je modelován jako proces s datovým záznamem základního typu PROCESS (reprezentován ukazatelem se jménem cpu). Proces probíhá podle programu CPU_PROG. Fronta na cpu je modelována jako objekt základního typu HEAD (reprezentace ukazatelem pfifocpu).

Periferní subsystémy jsou modelovány jako procesy s datovým záznamem typu PERIF, odvozeným ze základního typu PROCESS rozšířením o intenzitu obsluhy mi a ukazatel na frontu pfifo. Ukazatele na procesy jsou označeny jako per1, per2. Procesy periferních subsystémů využívají společný reentrantní program PERIF_PROG. Fronty na periferie jsou modelovány jako objekty základního typu HEAD (ukazatele pfifoper1, pfifoper2).

Aktivita plánovacího algoritmu operačního systému (plánovač – vybírá zakázky do multiprogramního mixu) je modelována pomocí procesu (datový záznam typu PROCESS, program SCHED_PROG, ukazatel sched). Proces pracuje nad frontou (typ HEAD, ukazatel pfifosched), do které je zařazena ukončená zakázka (pravděpodobnost větvení p0*i* po ukončení obsluhy v cpu). Plánovač je aktivován při každém vložení zakázky do fronty pfifosched a jeho činnost se liší podle aktuální hodnoty čítače cntsched. Hodnota čítače modeluje externí požadavky (zadávané z klávesnice) na zvýšení nebo snížení stupně multiprogramování (kladná nebo záporná hodnota čítače).

Externí požadavky jsou obsluhovány speciálním procesem (datový záznam typu `PROCESS`, program `KBD_PROG`, ukazatel `kbd`), který periodicky prohlíží (voláním funkce `scan_line()`) vyrovnávací paměť pro příkazy z klávesnice. Zadáním znaku `+` (a *Enter*) z klávesnice se inkrementuje hodnota čítače `cntsched`, zadáním znaku `-` a *Enter* se čítač dekrementuje. Při zjištění kteréhokoliv jiného počátečního znaku příkazu je volána funkce `stop()` a simulační výpočet se zastaví (je-li ovšem nastavena podmínka zastavení `Options/Break after/User's calling stop()`).

Činnost plánovače (tj. procesu `sched`) pro různé hodnoty čítače `cntsched` je následující:

`cntsched = 0` – plánovač náhodně změní typ prvního zakázky ve své frontě (`pfifosched`) voláním funkce `inijob()` a vrátí zakázku do obslužného systému `cpu`. Tímto způsobem je modelováno ukončení zakázky a jeho náhrada jiným při zachování konstantního počtu zakázek v systému.

`cntsched > 0` – kladná hodnota čítače indikuje požadavek na zvýšení počtu zakázek v systému. Plánovač provede stejnou akci jako v předchozím případě, vygeneruje novou zakázku, vloží ji do obslužného systému `cpu` a dekrementuje `cntsched`.

`cntsched < 0` – záporná hodnota čítače indikuje požadavek na zmenšení počtu zakázek v systému. Plánovač zruší první zakázku ze své fronty a inkrementuje `cntsched`.

Pro popisovaný způsob ovlivňování výpočtu zadáváním příkazů z klávesnice musí být aktivována příkazová řádka (volba `Options/Command line`).

Cílem simulace je určit hodnotu průchodnosti, tj. průměrný počet zakázek ukončených za jednotku času. Simulační model může být využit pro určení maximální hodnoty průchodnosti v závislosti na některém z parametrů modelu. Například lze určit optimální poměr uvažovaných dvou typů zakázek v multiprogramním mixu (tj. hodnotu pravděpodobnosti `q`, pro kterou je maximální průchodnost) pro dané pravděpodobnosti větvení (odpovídá četnost požadavků zakázek na periferní přenosy) a pro dané intenzity obsluh (souvisí s obslužnými časy v jednotlivých subsystémech obsluhy).

V inicializační sekci simulačního programu jsou definovány proměnné `micpu`, `miper1`, `miper2` a `q` jako parametry modelu pomocí funkce `def_par()`. Tyto veličiny lze měnit před každým simulačním experimentem pomocí

volby `Init` v hlavním ovládacím menu. Dále je v inicializační sekci definováno větší množství trasovaných proměnných a objektů pomocí funkce `def_trace()`. Aktuální hodnoty trasovaných veličin a stav objektů je možné zobrazovat v průběhu simulačního experimentu.

5.3 Modelování distribuovaných řídicích algoritmů

Využití eperimentální metodiky ověřování distribuovaných řídicích algoritmů předvedeme prostřednictvím poměrně univerzálního modelu takového algoritmu. Distribuovaným systémem zde rozumíme množinu procesorů vázaných komunikačním podsystémem, který zajišťuje *plnou konektivitu*, tj. možnost přímého komunikačního spojení mezi libovolnými dvěma procesory. Procesory komunikují zasíláním zpráv. Řídicím algoritmem rozumíme algoritmus řešící globální události týkající se funkce celého systému (výpadek procesoru, připojení nového procesoru, synchronizaci činnosti všech procesorů, volbu jednoho centrálního procesoru ap.).

Řidicí algoritmus je zpravidla koncipován jako programově realizovaný *konečný automat*, definovaný množinou vnitřních stavů $\{S_j\}$, množinou vstupů (zde typy zpráv $\{M_i\}$ akceptovaných automatem) a množinou operací (procedur) $\{O_{ij}\}$ prováděných při příchodu zprávy typu $\{M_i\}$ ve stavu $\{S_j\}$. Jednotlivé repliky - automaty v procesorech sítě jsou shodné (mohou se ovšem nacházet v různých vnitřních stavech) a vzájemně komunikují zasíláním zpráv (součást činnosti uvnitř $\{O_{ij}\}$). Ve složitějším asynchronním případě je doba přenosu zprávy náhodná. Jednotlivé procesory mají vstupní porty s vyrovnávací pamětí, která udržuje frontu došlých a automatem dosud nezpracovaných zpráv.

Model vstupního portu procesoru v **C-Sim** by mohl vypadat například takto:

```
typedef struct {
  d_head;          /* vlastnosti seznamu */
  PROCESS* server; /* obsluhující automat */
    ...            /* další položky */
} PORT;
```

Zprávy by měly tento typ:

```
typedef struct {
  d_link;          /* vlastnosti prvku seznamu */
  int code;        /* číselný kód zprávy */
    ...            /* další položky */
} MESSAGE;
```

Vlastní přechodová funkce automatu by byla univerzálně realizována jako matice ukazatelů na funkce operací, řádkový index je kód zprávy, sloupcový index je kód vnitřního stavu automatu. Pro jednoduchost zapíšeme jen matici 2 x 2 prvky:

```
void (* svitch_table [2][2]) () = {
  {O11 , O12},  /* např. O11 je jméno funkce vyvolané při  */
  {O21 , O22}   /*   příchodu zprávy s kódem 1 ve stavu 1  */
};
```

Data procesu realizujícího automat bychom popsali v **C-Sim** takto:

```
typedef struct {
  d_process;       /* vlastnosti pseudoparalelního procesu*/
  PORT* p_port;    /* vstupní port pro zprávy */
  int state;       /* vnitřní stav automatu */
    ...            /* další položky */
} P_DATA;
```

Dále naznačíme program procesu realizujícího automat. V **C-Sim**u mohou přístupový operátor k lokálním datům procesu (`my.`) používat pouze funkce, deklarované jako programy procesů (tj. pomocí makropříkazů `s_program` a `s_end_program`), tedy nikoliv funkce `Oij` realizující přechody automatu. Problém obejdeme zavedením globálního ukazatele `p_current`, který musí být před voláním funkce realizující přechod nastaven na aktivní proces a který může být využit pro přístup k lokálním datům (tj. `p_current ->` *položka* namísto `my.`*položka*). Reentrantní program, podle kterého probíhají procesy automatů ve všech procesorech, by potom vypadal zhruba takto

```
s_program  (P_DATA, P_PROG)
  my.state = výchozí_stav;
  while (1) {
    p_current = (P_DATA*)current();
```

```
      /* nastavení glob. ukazatele */
    if (!empty((HEAD*)my.p_port) {
      p_mes = (MESSAGE*) first ((HEAD*)my.p_port);
      hold (doba_přechodu);
        /* simulace doby zprac. zprávy */
      (*switch_table [p_mes->code] [my.state])();
        /* vlastní přechod  */
      out ((LINK*) p_mes);
        /* vyřazení zprávy z fronty */
    } else passivate();     /* žádná zpráva */
  }
s_end_program
```

Operace vyvolávané přes matici ukazatelů by byly realizované jako **C**-funkce. Naznačíme některou z nich:

```
void O11 () {
   ...
  p_current -> state = 2;
    /* změna stavu automatu      */
  send (message_poin, port_poin);
    /* zpráva jinému procesoru   */
   ...
}
```

Ve sdíleném adresáři určeném pro příklady z předmětu VSP je v souboru `D_EL.C` (zkratka *Distributed ELection*) model distribuovaného výpočetního systému (n procesorů a n replik řídicího procesu), ve kterém je realizován asynchronní algoritmus volby jednoho "vedoucího" procesoru (resp. procesu v něm realizovaného), přičemž některé procesory mohou být porouchané. V tomtéž adresáři je další komentář k příkladu.

Poznámka:

Prezentovaný způsob modelování činnosti distribuovaných a paralelních systémů umožňuje udržet semantickou blízkost modelované a reálně implementované verze řídicího algoritmu. Po odladění simulačního programu (tj. dlouhodobém stochastickém testování modelovaného algoritmu) lze přejít

od modelové k provozní verzi algoritmu relativně malými a průhlednými úpravami vesměs realizovatelnými systematicky pomocí editoru (či speciálního programu–filtru). Například se jedná o odstranění dynamiky modelu (viz příkaz `hold()`), systematické odstranění přístupových operátorů `my.` a `p_current->` a náhradu některých operací (zde např. `send()`, `passivate()`) voláním konkrétních funkcí reálného operačního prostředí procesu. Fakticky požadavek co největší blízkosti modelovaného distribuovaného či paralelního algoritmu (a možnost přímočarého využití algoritmu vyzkoušeného pomocí modelu) byl jedním z důvodů pro realizaci **C-Sim** jako extenze **C** jazyka, který je nejfrekventovanějším implementačnín prostředkem distribuovaných algoritmů. Detailnější informace o využití C-Sim pro modelování vlivu poruch na funkci řídicího systému lze nalézt na www stránce `http://www.c-sim.cz`, popřípadě `http://www.fit.zcu.cz`.

Literatura

[Dahl72] DAHL, O.J. - MYHRHAUG, B. - NYGARD, K.: SIMULA 67 Common Base Language. 2.ed. Oslo, Norsk Regnesentralen 1972.

[Dou81] DOUŠA, J.: Modelování na číslicových počítačích. Skriptum ČVUT. Praha 1981, 183 s.

[Dou90] DOUŠA, J.: Simulace. Skriptum pro postgraduální studium ČVUT. Praha 1990, 129 s.

[Češ94] ČEŠKA, M.: Petriho sítě. Brno, Akademické nakladatelství CERM 1994, 94 s.

[Havr86] HAVRDA, J.: Stochastické procesy a teorie informace. Skriptum ČVUT. Praha 1986, 276 s.

[Hlav89] HLAVIČKA, J.: Spolehlivost a diagnostika. Praha, Vydavatelství ČVUT 1989, 155 s.

[Hlav92] HLAVIČKA, J. - RACEK.S. - GOLAN.P. - BLAŽEK.T: Číslicové systémy odolné proti poruchám. Praha, Vydavatelství ČVUT 1992, 330 s.

[Hurt82] HURT, J.: Simulační metody. Skriptum UK. Praha 1982, 163 s.

[Kind80] KINDLER, E.: Simulační programovací jazyky. Praha, SNTL 1980, 277 s.

[Laub87] HUŠEK, R. - LAUBER.J: Simulační modely. Praha, SNTL 1987, 349 s.

[Mand85] MANDEL, P.: Pravděpodobnostní dynamické metody. Praha, Academia 1985, 181 s.

[Neu88] NEUSCHL, Š a kol.: Modelovanie a simulácia, Bratislava, Alfa 1988, 423 s.

[Oleh82] OLEHLA, M. - VĚCHET, V. - OLEHLA, J.: Řešení úloh matematické statistiky ve Fortranu. Praha, Nadas 1982, 363 s.

[PN90] ROZENBERG. G. (ed.): Lecture notes in Comp. Science 424 – Advances in Petri Nets 1989, Springer - Verlag 1990.

[Přib95] PŘIBYL, J.: Projektování a strategie distribuovaného zpracování dat. Praha, Vydavatelství ČVUT 1995, 231 s.

[Ráb82] RÁBOVÁ,Z - ČEŠKA,M.: Modelování a simulace. Skriptum VUT Brno, Praha, SNTL 1982, 337 s.

[Rac93] RACEK, S. - HEROUT, P.: C-Sim - Simulační nadstavba jazyka C. Plzeň, Tiskové a ediční středisko ZČU 1993, 38 s.

[Rou95] ROUBÍN, M.: C-Sim v.4.0. Diplomová práce. Plzeň, Katedra informatiky a výp. techniky ZČU 1995, 53 s.

[Ruk89] RUKOVANSKÝ, I.: Hodnocení výkonnosti počítačových systémů. Praha, SNTL 1989, 191 s.

[Siew82] SIEWIOREK, D.P. - SWARZ, R.S.: The Theory and Practice of Reliable System Design. Bedford, MA, Digital Press 1982, 772 s.

[Star91] STARÝ, I. - OBRUČA, L.: Teorie spolehlivosti. Skriptum Praha ČVUT, 1991, 171 s.

[Stau78] BENDA, Z. - STAUDEK, J.: Programování v jazyku SIMULA 67. Praha, SNTL 1978.

[Triv82] TRIVEDI, K.S.: Probability and Statistics with Reliability, Queuing and Computer Science Applications. Prentice-Hall 1982, 623 s.