IMS - Modelování a simulace

Studentská skripta, Akademický rok 2015/2016

Tento dokument vznikl jako skupinová práce studentů předmětu IMS na FIT VUT v Brně, v akademickém roce 2015/2016.

Obsah

Zdroje, Materiály

Originální příspěvek na Fitušce se zpracovanými tématy *(28.1. 2016 nezobrazuje vzorce)*
Záloha Fitušky 1, Záloha Fitušky 2 *(včetně vzorců)*
Shrnutí zadání půlsemestrálek a zkoušek
Sdílený dokument z 2014/2015
Runge Kutta 3. řádu, řešený
Euler - Easy

Teorie

Systém

- soubor elementárních částí (prvků systému), které mají mezi sebou určité vazby
- dvojice S = (U,R)

* U - universum, konečná množina prvků systému
* R - charakteristika, množina všech propojení prvků

Izomorfní systémy

- Prvky univerza U1 lze vzájemně jednoznačně (1:1) přiřadit prvkům univerza U2
- Prvky charakteristiky R1 lze vzájemně jednoznačně přiřadit prvkům charakteristiky R2, a to tak, že prvku charakteristiky R1, vyjadřujícímu orientovaný vztah mezi dvěma prvky univerza U1 , je vždy přiřazen právě ten prvek charakteristiky R2 , který vyjadřuje stejně orientovaný vztah mezi odpovídající dvojicí prvků univerza U2 a naopak.

Homomorfní systémy

- Prvkům univerza U1 je možno přiřadit jednoznačně prvky univerza U2 (opačně tomu tak být nemusí, N:1)
- Prvkům charakteristiky R1 je možno jednoznačně přiřadit prvky charakteristiky R2 , a to tak, že prvku charakteristiky R1 vyjadřujícímu orientovaný vztah mezi dvěma prvky univerza U1 je vždy přiřazen právě ten prvek charakteristiky R2, který vyjadřuje stejně orientovaný vztah mezi odpovídající dvojicí prvků univerza U2 ve smyslu bodu 1.

Model

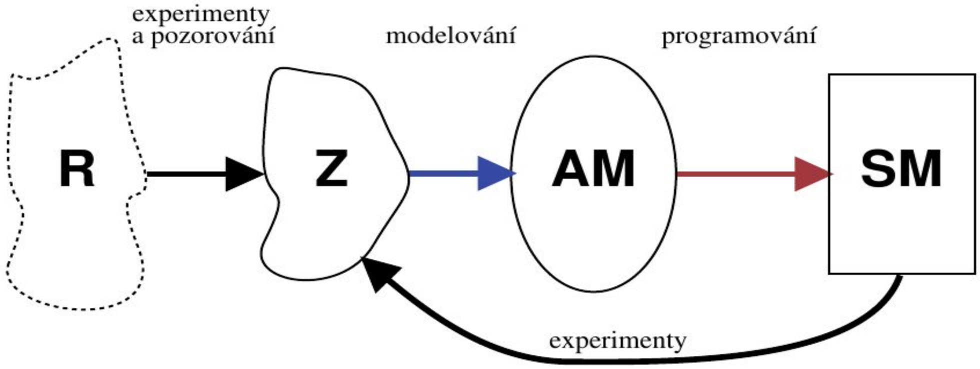
- napodobenina systému jiným systémem

Simulace

- získávání nových znalostí o systému experimentováním s jeho modelem

Postup při simulaci a modelování:

1. Vytvořím abstraktní model, zjednodušená realita
2. Vytvořím simulační model, naprogramovaný AM
3. Verifikace a validace
4. Simulace
5. Analýza a interpretace výsledků, nové znalosti



Reálný čas

- čas ve kterém probíhá skutečný děj v reálném systému

Modelový čas

- časová osa modelu, modeluje reálný čas z původního systému
  při simulaci nemusí být synchronní s časem reálným

Strojový čas

- je čas CPU spotřebovaný na výpočet programu (závisí na složitosti programu, počtu procesorů atd., nesouvisí přímo s modelovým časem)

Chování systému

- zobrazení množiny vstupů do systému na množinu jeho výstupů

Ekvivalence systémů

- 2 systémy jsou ekvivalentní, pokud stejné podněty vyvolají u obou stejné reakce
- dvojice podnětů/reakcí lze jednoznačně přiřadit definovaným vstup/výstupem

Verifikace

- ověření korespondence AM a SM (izomorfní vztah)

Validace

- ověření platnosti modelu, simulační model odpovídá realitě
- nelze určit absolutně, je to spíš jen míra platnosti (homomorfní vztah mezi R a SM)

Diskrétní rozdělení pst (pravděpodobnosti)

- fce rozdělení pst $pi = P(X = xi)$
- ● udává pst že hodnota nabývá dané hodnoty x
- ● suma hodnot = 1
- distribuční fce F(x) = P(X <= x)
- ● F(x) = suma pi od x1 do x

Spojité rozdělení pst

- fce hustoty pst f(x) = dF(x)/dx = F(x)’
- distribuční fce F(x) = P(X <= x) = integrál od -inf do x z f(x)dx

- na základě těchto definicí je třeba si uvědomit že například pokud máme fci hustoty pst s lineárním průběhem (např. f(x) = x) tak distribuční fce bude kvadratická (F(x) = x^2/2) a naopak

Generátory náhodných čísel
- náhodné - nedeterministické, fyzikální zdroje, například průchod radioaktivních částic apod., generují jen málo bitů za sekundu
- pseudonáhodné - deterministické, algoritmus pro generování, například kongruentní generátor, generují miliardy bitů za sekundu

SHO (Systém hromadné obsluhy)
- každý systém obsahující transakce (procesy) a popis jejich příchodů, obslužné linky a popis obsluhy, fronty různých typů ve kterých transakce čekají

Priorita procesu
- proces, který přijde k obsazenému zařízení se zařadí do prioritní fronty, předbíhá tak ostatní, kteří mají menší prioritu

Priorita obsluhy
- proces přijde k obsazené lince a vykopne toho, co tam zrovna je, vykopnutý jde buď čekat bokem, aby se posléze zase vrátil a mohl pokračovat v činnosti v zařízení, nebo začít znova, nebo odchází neobsloužen

Kendallova klasifikace SHO
- tvar: X/Y/c (je možné přidávat i další ale většinou je doplněna slovním popisem)
- X - typ stochastického procesu popisujícího příchod požadavků k obsluze
- Y - zákon rozložení délky obsluhy
- c - počet obslužných linek
- X a Y mohou být:
- M - Poissonův proces příchodů, neboli exponenciání rozložení
- Ek - Erlangovo rozložení
- D - pravidelný konstantní deterministický příchod
- G - jakékoliv *// stačí znát M a D.. jinak není třeba, sám PePe to prý taky neví*

Typy obslužných linek podle kapacity
- zařízení - kapacita = 1
- sklad - kapacita > 1
- zařízení obsahuje dvě fronty, vnější (požadavky čekají až dojdou na řadu) a vnitřní (přerušené požadavky čekají až zase budou moci pokračovat)

Událost
- diskrétní změna v systému, jednorázová nepřerušitelná akce vyvolaná v určitém modelovém čase

Proces

- posloupnost vzájemně souvisejících událostí

Klasifikace numerických metod

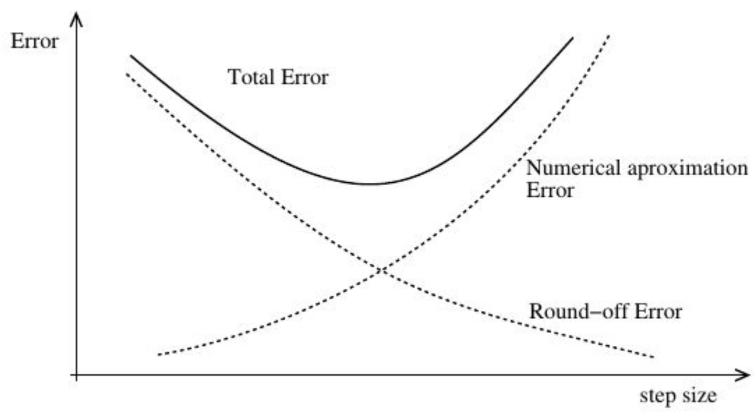
- jednokrokové - vycházejí pouze z aktuálního stavu
- vícekrokové - používají historii stavů/vstupů
- explicitní - výsledek dosazením do vzorce
- implicitní - v každém kroku musím řešit algebraické rovnice

Řád numerické integrační metody

- stupeň polynomu N (Taylorův rozvoj) definuje tzv. řád metody, tímto polynomem aproximujeme na základě aktuální hodnot

Chyby numerických metod

- lokální (v jednom kroku)
  - zaokrouhlovací (round-off) závislá na přesnosti aritmetiky daného zařízení, třeba float má tuto chybu větší než double
  - numerické aproximace (truncation) dána řádem metody - počtem členů Taylorova rozvoje pro výpočet další hodnoty, metody nižších řádů mají tuto chybu větší
- akumulované - sčítání vlivů lokálních chyb v průběhu výpočtu
- délka kroku ovlivňuje výrazně chyby, moc velký krok znamená nepřesnou numerickou aproximaci, moc malý zase nepřesné zaokrouhlování



Obrázek 6.9: Závislost chyby numerických metod na délce kroku

Eulerova metoda

- jednokroková, explicitní
- Runge-Kutta 1. řádu
- y(t+h) = y(t) + hf(t,y(t))

Runge-Kutta

- jednokroková, explicitní
- 2. řád
  k1 = hf(t,y(t))
  k2 = hf(t+h/2,y(t) + k1/2)
  y(t+h) = y(t) + k2

- **3. řád**
  k1 = h f(t, Y(t))
  k2 = h f(t+h/2, y(t) + k1/2)
  k3 = h f(t+h, y(t) - k1 + 2k2)
  y(t+h) = y(t) + k1/6 + k2 * 2/3 +k3/6
  https://en.wikipedia.org/wiki/List_of_Runge%E2%80%93Kutta_methods
  souhlas?

- 4. řád
  k1 = hf(t,Y(t))
  k2 = hf(t+h/2,y(t) + k1/2)
  k3 = hf(t+h/2,y(t) + k2/2)
  k4 = hf(t+h,y(t)+k3)
  y(t+h) = y(t) + k1/6 + k2/3 +k3/3 +k4/6

Tuhé systémy

- popis jejich chování zahrnuje jevy s výrazně rozdílnými časovými konstantami
- nelze vybrat optimální integrační krok dlouhý/krátký
- tuhost vyjadřuje koeficient tuhosti, který lze formulovat jako poměr nejrychlejších a nejpomalejších jevů systému
- je třeba použít speciální metody pro řešení

Algoritmus řízení spojité simulace

1. inicializace - nastavit počáteční stav
2. cyklus dokud není konec simulace
- pokud je vhodný čas - výstup
- vyhodnocení derivací a výpočet nového stavu
- posun modelového času
3. konec a výstup

Kombinovaná simulace
- spojitá + diskrétní + jejich spojení
- stavové události - nastávají při dosažení zadané hodnoty spojité veličiny (splnění stavové podmínky,nelze naplánovat), nemusí být detekována - nepřesné výpočty nebo moc dlouhý krok (přeskočíme)
- dokročení - úprava délky posledního kroku numerické metody, aby konec výpočtu vyšel na čas naplánované diskrétní události
- algoritmus, není celý, tato část se vkládá do next-event na místo, kde přiřazujeme čas: inicializace stavu a podmínek

```
Inicializace stavu a podmínek
while (čas < koncový čas) {
      if (čas + krok > koncový čas)         //dokroceni
            krok = koncovy čas - čas
      uložení stavu a času
      krok numerické integrace a posun času
      vyhodnocení podmínek
      if (podmínka změněna)
            if (krok <=minimální krok)
                  potvrzení změn podmínek
                  stavová událost
                  krok = běžná velikost kroku
            else
                  obnova stavu a času
                  krok = krok/2 // půlení intervalů
                  if (krok < minimální krok)
                        krok = minimální krok
}
```

Markovské procesy
- náhodné procesy splňující **Markovovu vlastnost**: následující stav procesu závisí jen na aktuální stavu

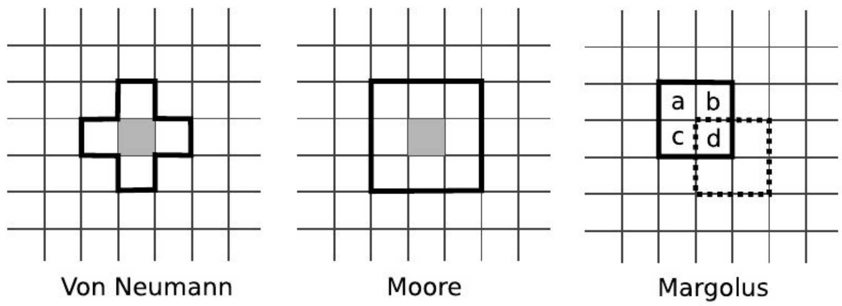
Markovský řetězec
- náhodný proces X(t) s diskrétním časem a diskrétními stavy, který má markovovu vlastnost

Spolehlivost
- schopnost plnit požadované funkce podle stanovených podmínek

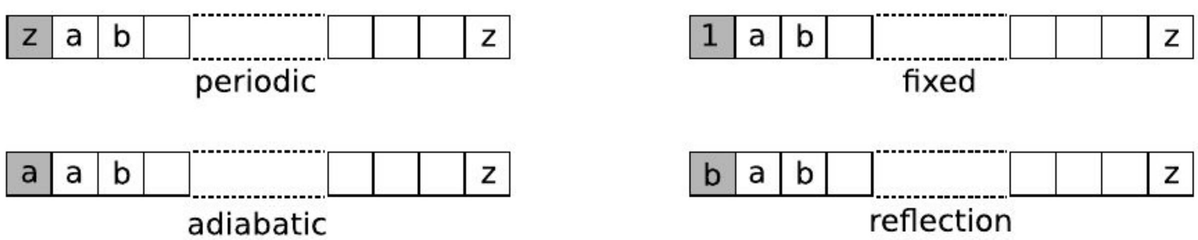
Celulární automat

- je typicky diskrétní systém obsahující:

- buňku - základní element, může být v jednom z konečného počtu stavů (např. {1,0})
- pole buněk: n-rozměrné (1D,2D...)
  - rovnoměrné rozdělení prostoru
  - může být konečné či nekonečné
- okolí - okolní buňky mající vliv na stav aktuální buňky
  typy:



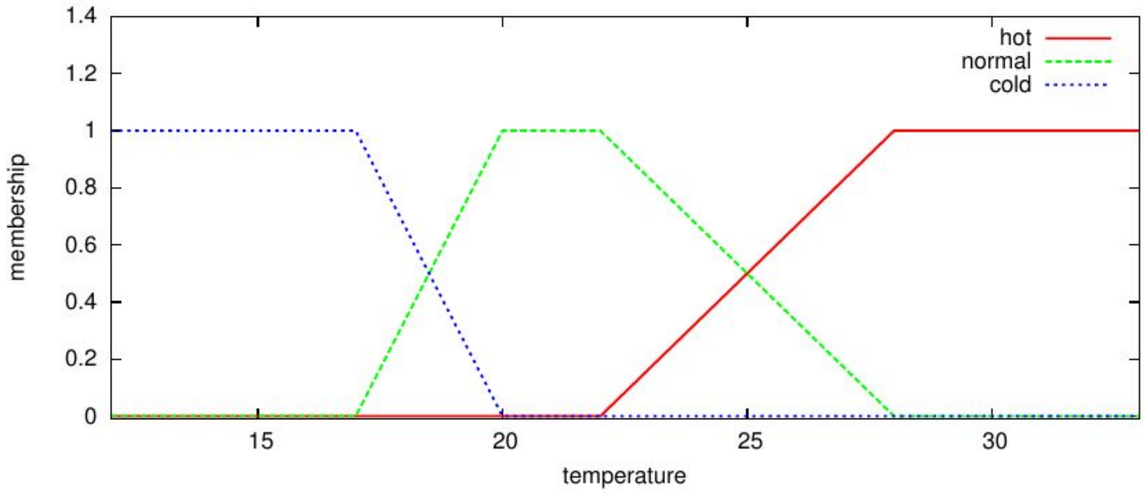
na okrajích pole je třeba aplikovat jednu z okrajových podmínek pro okolí



- pravidla - funkce stavu buňky a jejího okolí definující nový stav buňky

Fuzzy logika

- jde o popis jistého druhu neurčitosti/vágnosti (malé/velké apod.)
- jedná se o rozšíření Booleovské logiky
- hodnoty nejsou celé v určité množině ale je vyjádřena hodnota příslušnosti dané hodnoty do množiny



- mezi fuzzy operace patří: negace, konjunkce (min) a disjunkce (max)
- postup vyhodnocování:

1. převod vstupu na fuzzy množiny (fuzzifikace)
2. aplikace pravidel (if-then)
3. spojení výstupů pravidel (agregace)
4. převod zpět na ne-fuzzy hodnoty (defuzzifikace)

Co je perioda generátoru a jak byste ji zjistili experimentálně?

Perioda generátoru definuje za kolik vygenerovaných hodnot se začne generovat opět ta stejná posloupnost čísel znova.

Experimentálně je možné graficky si zobrazit vygenerované body a hledat periodu, což však nestačí, také je třeba provádět statistické testy.

Pro Tip: Já myslím, že chtěl slyšet "je potřeba provést i statistické testy".

Jaký modelový čas používají Markovské procesy?

Markovský proces -> spojitý čas

Markovuv retezec -> diskrétny čas

Jaký modelový čas používají metody systému hromadné obsluhy (M/M/1)?
Nenapadá mě žádný příklad SHO se spojitým časem, víte někdo o něčem takovém?

Pořadí vybírání záznamů z kalendáře
1. Podle aktivačního času
2. Podle priority
3. Podle pořadí naplánování události

Příklady

………

Popis a implementace kongruentního generátoru.

$x_{n+1} = (a*x_n + b) \mod m$

a a b konstanty, musí být vhodně definované

m také konstanta označující rozsah generátoru (nejvyšší číslo)

generuje rovnoměrné rozložení s konečnou posloupností (má periodu)

```
static unsigned long ix = seed;          //počáteční hodnota generátoru
double Random(void) {
      ix = ix * 69069L + 1;              //vhodně zvolené konstanty, m je zde
implicitně daná architekturou - velikostí double 32bit - 2^32 (předělení mod vlastně udělá C
samo tím že rotuje hodnotu od 0 po překročení max hodnoty)
      return ix / ((double)ULONG_MAX + 1); //normalizace na  <0,1)
}
```

………

Metoda inverzní transformace na exponenciální rozložení.

1. Inverze distribuční fce
   (kdyby někdo nevěděl co je to inverze, tak všechny x si přepíšu na y a opačně a pak si z takto upraveného vzorce vyjádřím y)
   F(x) = 1 - e^-((x-x0)/A) =>
   x = 1 - e^((x0-y)/A)
   e^((x0-y)/A) = 1-x | zlogaritmuju (exponent můžu vyhodit před ln)
   ((x0-y)/A)ln(e) = ln(1-x)
   y = x0 - A*ln(1-x)
2. Za x vygenerování Uniform(0,1)
3. Výsledek: y=F^-1(x)

POZOR: Když má v příkladu člověk napsat random funkci se zadaným rozložením, nejdřív dvakrát zkontroluju, že to nejde dělat vylučovací metodou (tj. nemůžu kolem funkce hustoty udělat obdélník, je nekonečná). U vylučovací metody totiž rovnou píšu zdroják, zatímco u inverzní transformace si člověk nejdřív pěkně započítá - viz příklad a poznámka u vylučovací metody.

………………………………………………………………………………………………………

Vylučovací metoda a transformace na jiné rozložení.

Náhodná veličina má fci hustoty f(x), x náleží <x1,x2) a f(x) náleží <0,M>

1. x = Uniform(x1,x2)
2. y = Uniform(0,M)
3. if y < f(x) then x je hodnota hledané náhodné veličiny else goto 1

Příklad z roku 2012/13:

Napište funkci MyRand() podle rozložení, použijte vhodnou metodu a popište co dělá, porovnejte s jinou metodou:

f(x) = x pro interval <0, 1)
f(x) = 2-x pro interval <1,2)
f(x) = 0 pro zbytek

Moje řešení, histogram sedí, **potvrzeno 2krát**:

```
double MyRand() {
  while (1) {
    double x = Random() * 2; //proč *2? x je v rozsahu <0;2)
    double y = Random();
    if ((x < 1 && y < x) || (x >= 1 && y < 2 - x)) {
      return x;
    }
  }
}
```

Pozn.: Příklad lze řešit také *inverzní transformací* (pokoušeli se o to na Fitušce), ale připravte se, že si započítáte, když to jde jó dobře, 20minut je v háji:

> Moje řešení, histogram sedí, http://www.imagehosting.cz/?v=inverznime.jpg

……

Příklad z 1. termínu na kongruentní generátor.

Mějme funkci hustoty pravděpodobnosti f(x) = ¼ pro x € <0,1) a f(x) = ¾ pro x € <1,2), jinak f(x) = 0

Nakreslete graf průběhu f(x), distribuční funkce F(x) tohoto rozložení (přesně vyznačit minimálně 3 body průběhu) a průběh inverzní distribuční funkce F^-1 (x).

a) *V C naprogramujte generátor tohoto rozložení double gen1() s využitím metody vylučovací, použijte funkci Random()*

```
double gen1()
{
        while(1)
        {
                double x = Random()*2;
                double y = Random()*3/4; // proč *¾? je to ta hodnota ze zadání -> y
(nejvyšší hodnota v oboru hodnot) = nad funkcí sestrojujeme co nejmenší obdélník

                if ((x<1 && y<1/4) || (x<2 && x>=1))
                        return x;
        }
}
```

b) *double gen2() - to samé, akorát s využitím inverzní transformace*

"Distribuční funkce je neklesající, zprava spojitá, její limita $-\infty$ je nula, v ∞ pak jedna."

převést hustotu na distribuční fci

F(x) = integrál z f(x)

při integraci vychází vždy hodnota + konstanta, kterou vypočítáme dosazením

první rovnice: f(x) = ¼ pro x € <0,1)

integrál z ¼ je tedy (1/4)*x + C

víme že v bodě 0 je F(x) = 0, proto

(¼)*0 + C = 0 => C=0

<u>1/4 * x pro (0;1></u>

druhá rovnice: f(x) = ¾ pro x € <1,2)

integrál je (¾) * x + C

víme že bod 2 je maximální v naší fci takže F(2) = 1

(¾)*2 + C = 1 => C = -½

<u>(¾) * x - 1/2 pro (1;2></u>

<u>výpočet invertované distribuční funkce:</u>

y = 3x/4 - ½   <= zamenime x a y (inverzia) a vyjadrime y

x = 3y/4 - ½

x + ½ = 3y/4

y = 4x/3 + ½ / ¾

y = 4x/3 + ⅔

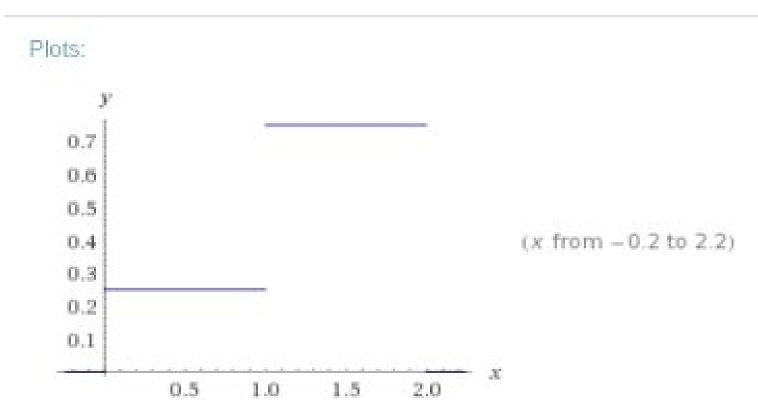
<u>inverzní fce jsou tedy:</u>
y = 4x pro x <0,1/4)
y = 4/3* x + ⅔ pro x <¼,1)

<u>vysledny kod je tedy:</u>

```
double gen2() {
        x = Random();
        if (x<1/4) // pred zlomem
                return(4 * x);
        else // random vraci do 1), neni potreba osetrovat konec
                return(4/3 * x + 2/3);
}
```

<u>Grafy:</u>
<u>Původní fce:</u>

$$\begin{cases} \frac{1}{4} & x \geq 0 \wedge x < 1 \\ \frac{3}{4} & x \geq 1 \wedge x < 2 \end{cases}$$

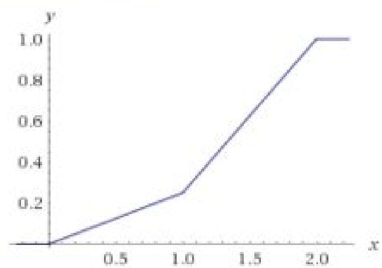


<u>Distribuční fce:</u>

$$\int \left(\begin{cases} \frac{1}{4} & x \geq 0 \wedge x < 1 \\ \frac{3}{4} & x \geq 1 \wedge x < 2 \end{cases} \right) dx = \begin{cases} 0 & x \leq 0 \\ \frac{x}{4} & 0 < x \leq 1 \\ \frac{3x}{4} - \frac{1}{2} & 1 < x \leq 2 \\ 1 & \text{(otherwise)} \end{cases} + \text{constant}$$

$e_1 \wedge e_2 \wedge \ldots$ is

Plots of the integral:

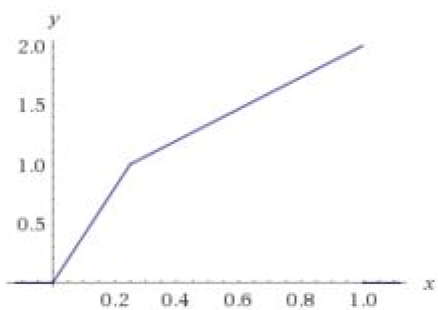


(x from -0.2 to 2.2)

Inverzní k F(x):

$$\begin{cases} 4x & x \geq 0 \bigwedge x < \frac{1}{4} \\ \frac{4x}{3} + \frac{2}{3} & x \geq \frac{1}{4} \bigwedge x < 1 \end{cases}$$

Plots:



(x from -0.1 to 1.1)

....

Monte Carlo, popis a aplikace na integrál fce sinus.

1. Vytvořím stochastický model
2. Provedu náhodné experimenty
3. Získanou pst nebo průměr použiju k výpočtu výsledku

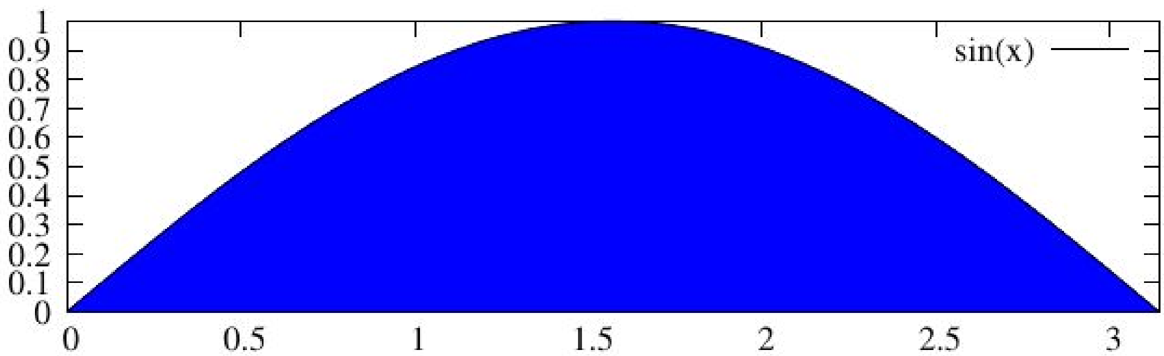
Přesnost metody: err = 1/sqrt(N)

N - počet provedených pokusů

integrál od 0 do pí z fce sin(x)dx

1. Vygeneruju N náhodných bodů [xi,yi] v rozsahu x náleží <0,pí> a y náleží <0,1>
2. Vypočtu pst jevu yi < sin(xi) (bod leží v ploše pod fcí sinus - definice integrálu)

3. Výsledek je přibližně = |rozsah x| * |rozsah y| * P (celková plocha krát pst že jsem se trefil = obsah plochy pod fcí sin, tedy integrál)



<u>Rozepsané více:</u>

```
double y_range = 1;
double x_range = 3.14;

int getVal() {
	double x = Random() * x_range;
	double y = Random() * y_range;

	return y < sin(x);
}
```

<u>Potom nějaky wrapper nad tou metodou, podle kterého vypočte pravděpodobnost</u>

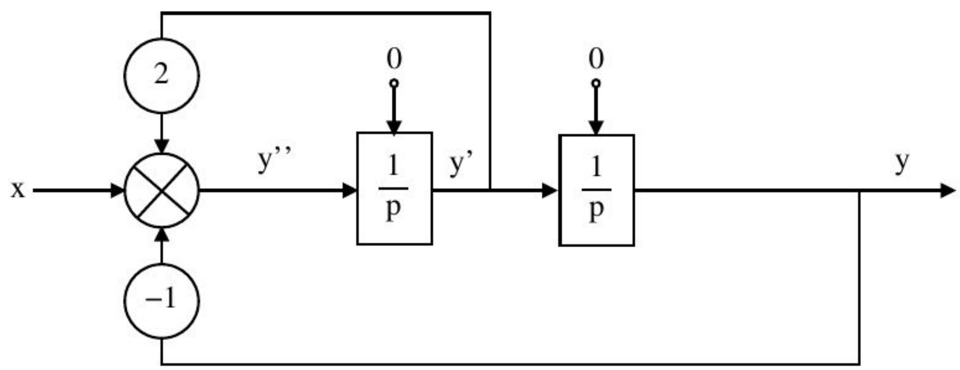
```
double monteCarlo() {
	int attempts = 15; // pocet pokusu
	int hits = 0;
	for (int i = 0; i < attempts; i++) {
		if (getVal()) {
			hits++;
		}
	}
	return (hits/attempts) * y_range * x_range;
}
```

………………………………………………………………………………………………………

Snižování řádu derivace pro y’’ -2y’ + y = x

Tuto metodu lze použít pouze pokud nemám derivace vstupů (x’, x’’ atd)!!!!

1. Osamostatním nejvyšší řád derivace
   y’’ = 2y’ - y + x
2. Snažím se zapojit několik integrátorů za sebe a vzájemně je propojit, abych se zbavil derivací
   y’ = ∫ y’’
   y = ∫ y’



………………………………………………………………………………………………………

Metoda postupné integrace pro p^2y + 2py + y = p^2x + 3px + 2x

Vhodná když máme derivace vstupů. To p značí derivaci, prostě jiný zápis, stejně tak 1/p je integrál. (je to laplaceova transformace btw)

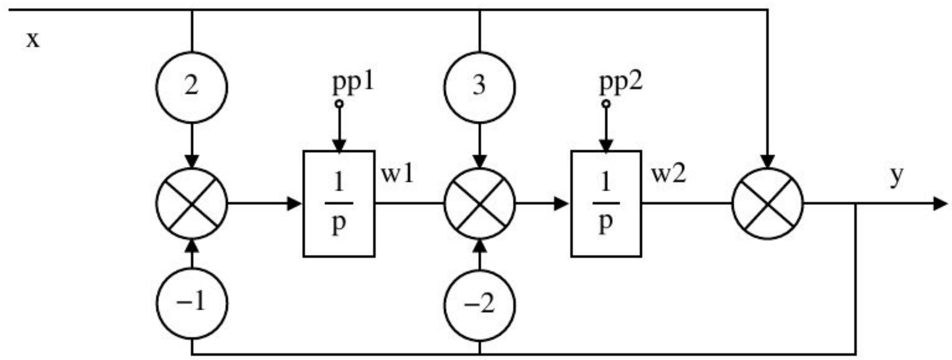
1. Osamostatním nejvyšší derivaci
   p můžu vytýkat před závorky
   p^2y = p^2x + p(3x - 2y) + 2x - y
2. Postupně integruji a hodnoty s integrálem ukládám do nových proměnných (substituce) snižuji tak řád derivace
   py = px + (3x - 2y) + 1/p(2x-y)
   w1 = 1/p(2x-y)

   py = px + (3x - 2y) + w1
   y = x + 1/p(3x - 2y + w1)
   w2 = 1/p(3x - 2y + w1)

   y = x + w2
3. Výpočet nových počátečních podmínek
   w2(0) = y(0) - x(0) (z poslední rovnice)

   y = x + 1/p(3x -2y + w1)
   zderivujeme * p

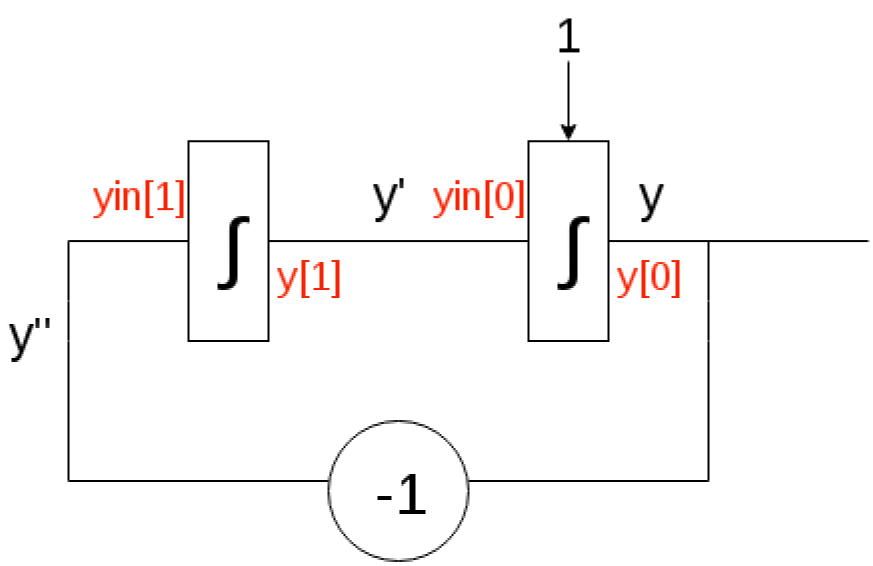
py = px + 3x - 2y + w1
w1 = py - px - 3x + 2y
w1 = py(0) - px(0) - 3x(0) + 2y(0)



……

Příklad implementace numerické metody v jazyce C.
Rovnice: y’’ + y = 0
Metoda: Eulerova
Počáteční podmínky: y(0) = 1

Nakreslím si blokové schéma, nebo si jinak ujasním jak jsou spojené integrátory:
y’’ = -y



Červené části jsou pro ilustraci kódu níže

Definuju si vstupy yin a výstupy y integrátorů s tím že do výstupů zadám počáteční podmínky (pokud nejsou zadané tak implicitně 0, viz y[0])

```
double yin[2];             // vstupy integrátorů
double  y[2] = { 0.0, 1.0 };     // počáteční podmínky
double  time = 0;          // modelový čas
double h = 0.01;           // krok numerické metody
```

Implementace má standardně tři funkce:
Fce dynamic (někdy update) řeší spojení mezi integrátory, co je na vstupech a v každém kroku poskytne nové hodnoty
Jinými slovy se zde počítá f ze vzorce pro Eulera
(pokud to dáte přímo do cyklu níže tak se podle slov pana Dr. Peringera střelíte do nohy, mohlo by se podělat pořadí vyhodnocení vstupů, vyhodnocení vstupů musí být vždy odděleno od vyhodnocení výstupů!!!)

```
void dynamic() {      // Popis systému: výpočet VSTUPŮ integrátorů
      yin[0] = y[1];        // y'
      yin[1] = -y[0]; // y'' = -y
}
```

Zde se provádí krok Eulerovy metody
y(t + h) = y (t) + hf (t, y(t)), tento vzorec je implementován v cyklu for níže, yin už víme že je vlastně to f (viz dynamic)

Provedeme pro všechny integrátory

```
void integrate_euler() {         // krok integrace:
      dynamic();                 // výpočet aktuálních vstupů
      for (int i = 0; i < 2; i++)// postupně pro všechny integrátory
            y[i] += h * yin[i];  // výpočet nového stavu
      time += h;                 // posun času
}
```

Zde je aplikován algoritmus řízení spojité simulace, na začátku je možná inicializace, kterou v našem případě máme již nahoře v globálu, zde dělám prostě kroky dané metody a vypisuju si je (podobně jako minulý rok v INM ty tabulky s hodnotama výsledků numerické metody)

```
int main() {          // popis experimentu
      while (time < 20) {
            printf("%10f %10f\n", time, y[0]);
            integrate_euler();
      }
      return 0;
}
```

Více exkluzivně v přednášce 3.11.2015, IMS.pdf (str. 225), opora-ims.pdf (str. 65)

………………………………………………… ……………………………………………………

Runge-kutta z 1. termínu

Napište funkci pro výpočet jednoho kroku ***`RK_step(double t, double state[], unsigned N, double stepsize)`****, které předáte modelový čas t, stav všech integrátorů v 4poli state, jejich počet N a délku kroku.*

Chování modelu je popsáno funkcí ***`Dynamic(double t, double st[], unsigned N, /* out */ double in[]);`***
ve které se vypočítají vstupy (in[i]) všech integrátorů pro dané stavy (st[i]) a aktuální modelový čas t. Definujte ji konkrétně pro rovnici ***`y''' + 7y' - 5 = 0`***
A nastavte správnou hodnotu konstanty N. Doplňte počáteční nenulové podmínky.
Doplňte vše potřebné ve funkci ***void SimulationRun(double t1, double t2) s využitím RK_step****. Nezapomeňte dokročit na čas konce simulace t2.*

RK4: 4. řád

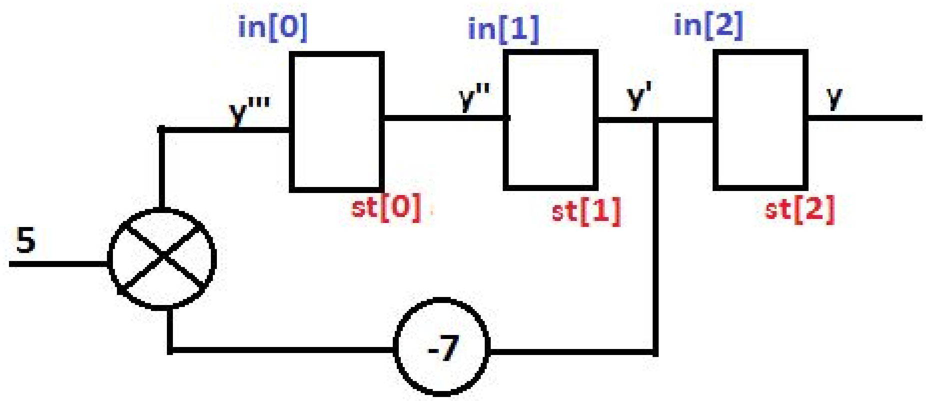
$$k_1 = hf(t, y(t))$$
$$k_2 = hf(t + \frac{h}{2}, y(t) + \frac{k_1}{2})$$
$$k_3 = hf(t + \frac{h}{2}, y(t) + \frac{k_2}{2})$$
$$k_4 = hf(t + h, y(t) + k_3)$$
$$y(t+h) = y(t) + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}$$

Příklad z materiálů k předmětu na RK4 a kruhový test:
https://www.fit.vutbr.cz/study/courses/IMS/public/priklady/rk4-test.c.html

Upravená verze kruhového testu pro toto zadání (je potřeba jej zkontrolovat - nemusí být úplně korektně): https://gist.github.com/JarekParal/aaeb1d079156107694e59368e2f0147f

Podobný příklad na RK::
https://docs.google.com/document/d/1IK_j0PZWfrDojHAjQC38rbV4I55xiWAsgYx51DVW4ak/edit

y''' + 7y' - 5 = 0   =>   y''' = 5 - 7y'


in[0]
in[1]
in[2]
y'''
y''
y'
y
st[0]
st[1]
st[2]
5
-7

dynamic (double t, double st[], unsigned N, double in[])

```
{
    in[0] = -7 * st[1] + 5;
    in[1] = st[0];
    in[2] = st[1];
}
```

RK_step(double t, double state[], unsigned N, double stepsize)

```
{
    double start[N];
    double in[N];
    double k1[N];
    double k2[N];
    double k3[N];
    double k4[N];
    int i;

    for (i = 0; i < N; i++)
            start[i] = state[i]; //ulozeni pocatecniho stavu

    dynamic(t, state, N, in); //vypocet novych stavu

    for (i = 0; i < N; i++)
    {
            k1[i] = stepsize*in[i];  //vypocet k1 podle vzorce h*f(t,y(t)) => stepsize*in[i]
            state[i] = start[i] + k1[i] / 2; //vypocet vstupniho stavu podle vzorce
    }
    dynamic(t + stepsize / 2, state, N, in); //vypocet novych  stavu

           //toto je potreba udelat pro kazde k

    for (i = 0; i < N; i++)
    {
            k2[i] = stepsize*in[i];
           state[i] = start[i] + k2[i] / 2;
    }
    dynamic(t + stepsize / 2, state, N, in);

    for (i = 0; i < N; i++)
    {
            k3[i] = stepsize*in[i];
            state[i] = start[i] + k3[i];
    }
```

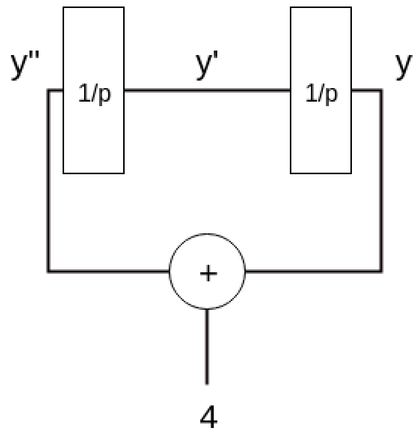
```
    dynamic(t + stepsize, state, N, in);

    for (i = 0; i < N; i++)
    {
            k4[i] = stepsize*in[i];
            state[i] = start[i] + k1[i]/6 + k2[i]/3 + k3[i]/3 + k4[i]/6; //vysledek
    }

          // dynamic(t, state, N, in); //tady dělá f(t,y)
          //
          // for (i = 0; i < N; i++)
          // {
          //        k1[i] = stepsize*in[i]; //toto je k1=h*f
          //        state[i] = start[i] + k1/2; //a toto už jakoby patří do dalšího výpočtu toto už je
          //        //příprava pro f(t+h/2, y+k1/2)
          // }
          // //tady pak dělá tu f a tak dále...
          // dynamic(t + stepsize / 2, st, N, in); //vypocet novych  stavu
}

void SimulationRun(double t1, double t2)
{
    int N = 3;     //tri integratory
    double state[N] = {0.0, 0.0, 1.0} //pocatecni podminky -> jen priklad, nejspis je potreba je
                                        //dopocitat
    double stepsize = 0.1;

    while (t1 < t2){
            printf("some rubbish");
            RK_step(t1, state, N, stepsize);
            if (t1 + stepsize > t2){
                    t1 = t2 - stepsize;  //dokroceni na konec simulace
            }
            t1 += stepsize;
    }
    printf(#Result);
}
```

……

y'' - y - 4 = 0, převést na soustavu rovnic 1. řádu a podle algoritmu řízení spojité simulace spočítat Eulerem do času 1.5

h = 0.5, y'(0) = 0, y(0) = -2

y'' = y+4



Postup je vlastně převedení programu z příkladu výše (implementace Eulera) do praxe ujasním si vstupy a výstupy integrátorů a použiju algoritmus

1. **vyhodnocení integrátorů (dynamic aka update)**
2. **euler**
3. **posun času**

y = y + h*f(t,y(t)) převedu na:

y = y + h*yin **kde yin reprezentuje vstup integrátoru a vyhodnocení fce f**

ze zadaných poč podmínek a upravené rovnice ze zadání vypočítám y'' = y + 4 = -2+4 = 2

a teď jedu podle algoritmu

y = -2 + 0.5*0 = -2 -2 **je původní hodnota y (v t=0) a 0 je y', tedy yin (viz obrázek výše)**

y' = 0 + 0.5*2 = 1

y'' = -2+4 = 2

…

Trosku klarifikace (mně to nebylo na prvni pohled uplne jasne)

mam uz hodnotu y'(0) i y(0) (y''(0) logicky nemam)

pro cas 0 mi jeste chybi y'' - z nacrtku vidim, ze to je vlastne y + 4 = 2

jedu dal, pro cas 0.5 : (ted uz mam vsechny defaultni hodnoty, takze proste pojedu po dratu)

y' = 0 (hodnota z minulosti) + 0.5 (krok) * 2 (input - vystup z y'') = 1

y = -2 (minulost) + 0.5 (krok) * 0 (input - vystup z y') = -2

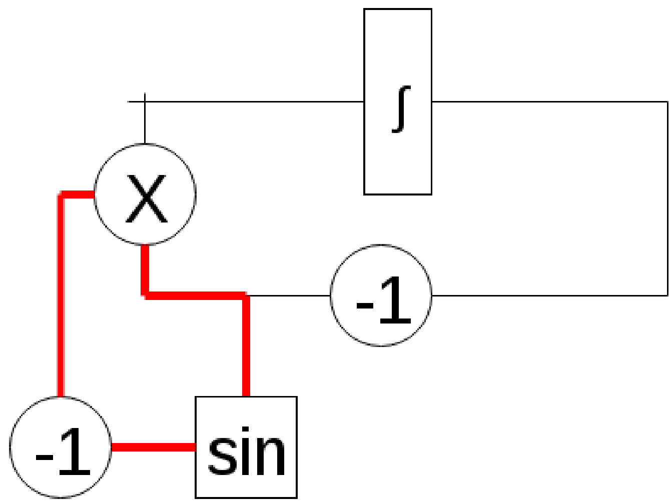
~~y'' = 2 (minulost) + 0.5 (krok) * 2 (input - 4 + vystup z y) = 3~~

t ->	0	0.5	1	1.5
y	-2	-2	-1.5	-0.5
y’	0	1	2	3.25
y’’	2	2	2.5	3.5

…………………………………………………………………………………………………….

Řazení funkčních bloků a detekce rychlých smyček

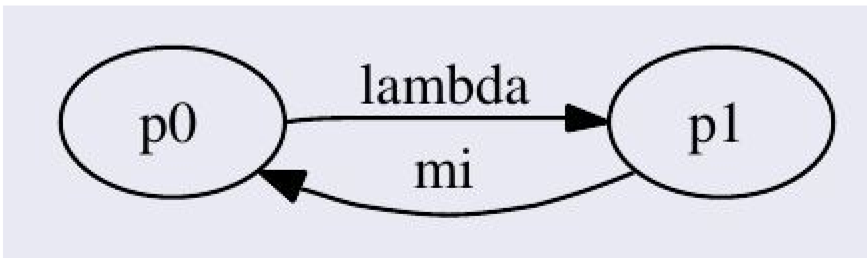
Bloky se stavem jako jsou integrátory nebo zpoždění vždy narušují rychlou smyčku. Postup je tedy zakrýt si integrátory či jiné stavové prvky a hledat cyklus u ostatních algebraických bloků.



Řešením rychlých smyček je a) přepracování modelu, b) vložení speciálního bloku, který řeší algebraické rovnice.

……
Máme jedno zařízení bez fronty, přijde požadavek a nemůže být obsloužen, odchází.
Příchody: 15 za hodinu
Obsluha: 5 minut

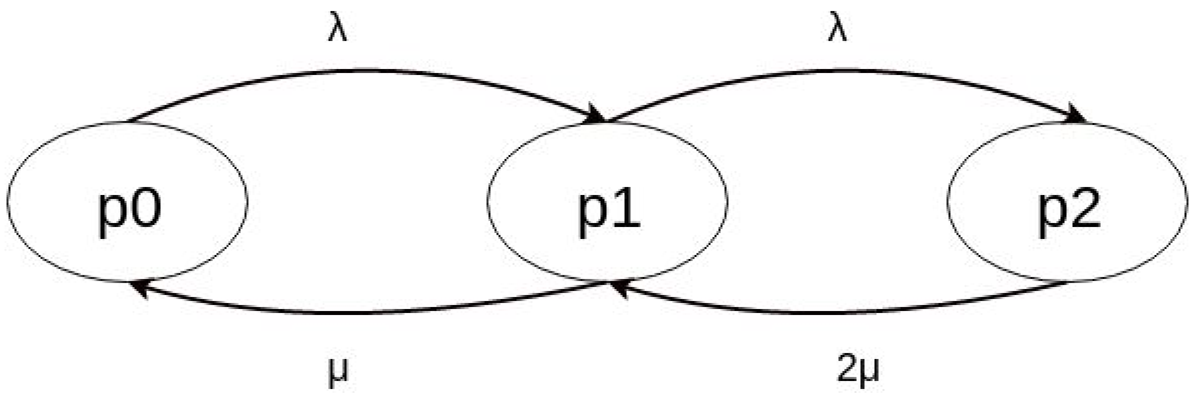
Jaká je pst že požadavek odejde neobsloužen?
$\lambda = 15$, $\mu = 60/5 = 12$ za hodinu



p0 - pst že zařízení je prázdné, s intenzitou mí jdu ze stavu obsazeno do stavu volné
p1 - pst že obsazené, do stavu s pst p1 jdu s intenzitou lambda
na začátku je p0 = 1 a p1 = 0
pak se mění a po nějakém čase (v nekonečnu třeba) se pst ustálí
chceme p1 a p0 v ustáleném stavu, tam se pst nemění -> jejich derivace = 0 -> z diferenciálních rovnic se stanou algebraické
$p0 + p1 = 1$ **jejich součet musí být 1, jsou to pst**
$-\lambda p0 + \mu p1 = 0$ **sestaveno vzhledem k p0, jinými slovy co odteče z p0 (**$-\lambda p0$ **) to tam zase i přiteče zpátky (**$\mu p1$**)**
$-\lambda(1 - p1) + \mu p1 = 0$
$p1 = \lambda/(\lambda+\mu)$
$p1 = 5/9$ - pst obsazeného zařízení

Pozn.: systém s neomezenou frontou je řešitelný jen tehdy, je-li stabilní
Pro systémy M/M/1 je podmínka stability $\lambda < \mu$
Pro M/M/2 je podmínka $\lambda < 2\mu$ (zpracováváme 2x více požadavků - 2 zař.), apod.
Obecně pro M/M/x je podm. $\lambda < x\mu$

……
Systém M/M/2, μ = 4, λ=3, jaká je pst že jsou obě zařízení prázdná?



p0+p1+p2=1
-λp0 + μp1 = 0
=> p1 = (λp0)/μ

λp1 - 2μp2 = 0
=> p2 = (λp1)/(2μ) => p2 = ((λ^2)p0)/(2μ^2)

p0 = 1-p1-p2 => p0 = 1-(λp0)/μ-((λ^2)p0)/(2μ^2)
p0=**32/65 je to OK**

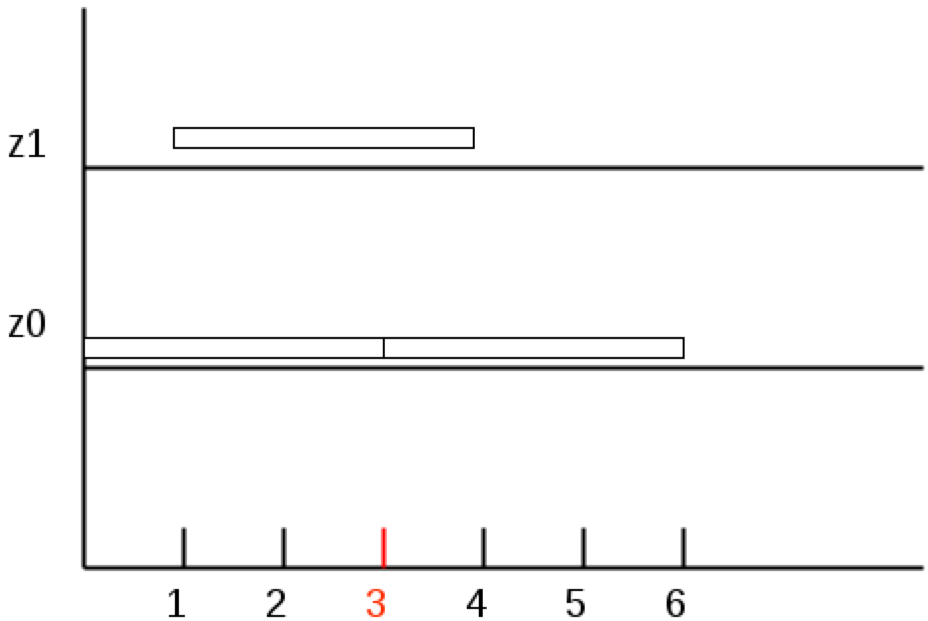
……
Jaký je celkový počet všech možných pravidel u celulárního automatu pro 1D okolí. Buňky mají stavy 1 nebo 0 (elementární CA).

Na vstupu máme 3 buňky, aktuální + dvě okolo.
Existuje 2^3 = 8 možností vstupů (vstupy 0/1) => 2^8 možných pravidel.

……
Popis algoritmu next-event a demonstrace činnosti
Algortimus:
inicializace času, kalendáře, modelu…
```
while (kalendář.empty() == False) {
      vezmi první záznam z kalendáře
      if (aktivační čas události > koncový čas)
            konec simulace
      nastav čas simulace na aktivační čas události
      proveď popis chování události
}
```
Záznam kalendáře obsahuje vždy aktivační čas, prioritu a popis činnosti události. Události samotné nejsou uložené v kalendáři!!!

Demonstrace:

Model systému D/D/2, společná LIFO fronta, příchody požadavků začínají v čase 0, interval příchodů 1s, doba obsluhy 3s, čas od 0 do 99s, zařízení z0 má prioritu před z1. Priority transakcí neuvažujeme. Otázka je například popsat situaci a obsah kalendáře v určitém čase.



Časový diagram:

Poznámka: u LIFO fronty pak beru vždycky poslední transakci ve frontě, u FIFO zase první (tu která je tam nejdéle)

Čas 0:
- přijde první požadavek p0 a obsadí z0, naplánuje uvolnění v t+3
- naplánuje se příchod p1 v čase t+1

Čas 1:
- přijde p1 a obsadí z1, -||-
- naplánuje se příchod p2 v čase t+1

Čas 2:
- přijde p2, jde čekat do fronty
- naplánuje se příchod p3 v čase t+1

Čas 3:
- první v kalendáři je uvolnění z0 (mám stejné priority a stejné časy u uvolnění z0 i u příchodu p3, takovém případě beru záznamy tak, jak byly vloženy do kalendáře, první bylo uvolnění)
- fronta je neprázdná -> naplánuju obsazení z0 v čase t transakcí p2 (plánuju na aktuální čas)
- přijde p3, naplánuje obsazení z0

- p2 obsazuje z0
- p3 chce obsadit, je plno -> jde do fronty

….

………

Sestavit blokové schéma:

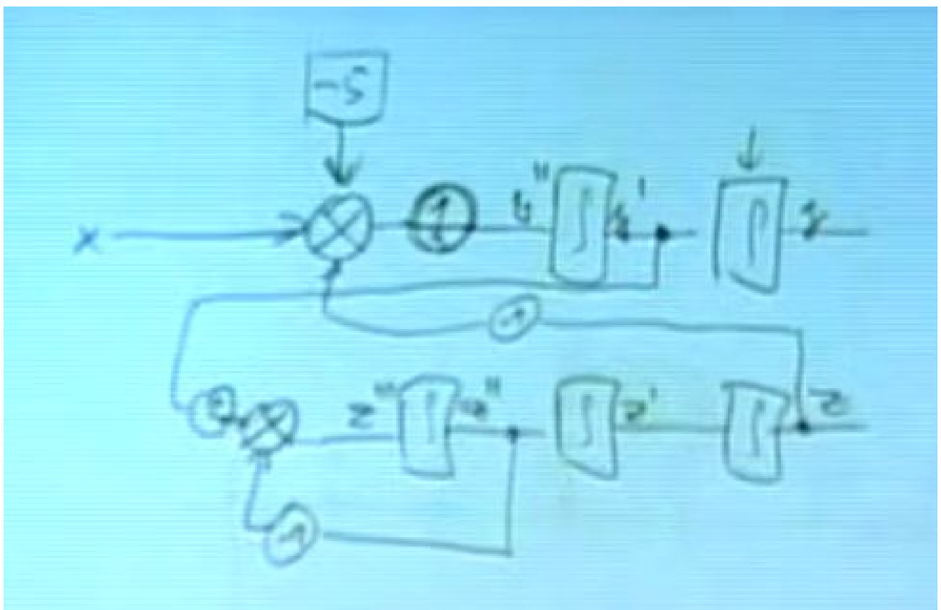
$3y'' + z + 5 = x$

$z''' + z'' - 2y' = 0$

Použijeme metodu snižování řádu derivace.

$y'' = (x-5-z)/3$

$z''' = 2y' - z''$



………

Napiště pseudokód (max. 5 řádků s využitím SIMLIB/C++) popisující část chování procesu, který obsadí zařízení s přerušením probíhající obsluhy.

Priorita obsluhy (slide 183):

```
void Behavior() {
  Seize(Fac,1);
  Wait(...);
  Release(Fac);
}
```

Napiště pseudokód (max. 5 řádků s využitím SIMLIB/C++) popisující část chování procesu, který se pokouší o obsazení zařízení s předbíháním ve vstupní frontě.

Priorita procesu (slide 178):

```
void Behavior() {
   priority = 3; // nastavíme procesu vyšší prioritu, předběhne frontu
   Seize(Fac);
   priority = 0; // vrátíme prioritu na původní úroveň
}
```

Napište v jazyku C funkci `double mcintegral()` pro výpočet N-násobného určitého integrálu funkce `double f(double x[], unsigned N)` (podle všech x_i, $i \in \{0,... N-1\}$ metodou Monte Carlo, s požadovanou přesností přibližně 1%. Meze integrálu jsou uloženy v globálním poli struktur `Rozsah` s položkami `.min` a `.max`.
Předpokládejte, že neznáte rozsah funkčních hodnot funkce `f`.
Mate k dispozici pouze funkci `double Random(void)` vracející pseudonáhodné číslo v rozsahu <0,1), `f` a žádnou jinou.

```
double precision = 0.01;

double mcintergral() {
      double sum=0;
      double products=1;
      int iterations=(int) (1/(precision*precision));
      for(unsigned i=0;i<N;i++) {
            products*=(Rozsah[i].max-Rozsah[i].min);
      }
      for(unsigned i=0;i<iterations;i++) {
            double vect[N];
            for(unsigned o=0;o<N;o++) {
                  vect[o]=(Random()*(Rozsah[o].max-Rozsah[o].min))
                           +Rozsah[o].min;
            }
            sum+=f(vect,N);
      }
      double average=sum/iterations;
      return average*products;
```

```
}
```

OTÁZKY

1. (15 bodů) Napište pseudokód algoritmu řízení diskrétní simulace typu *"next-event"* (nesmíte použít speciální ukončovací událost). *POZOR: bude-li chybět, -5b.*

   Mějme model systému typu D/D/3 se společnou LIFO frontou. Příchody požadavků začínají v čase 0 s intervalem $1s$ a doba obsluhy trvá $5s$. Simulace bude provedena pro čas od 0 do $99s$. Požadavky číslujte od 0 podle pořadí příchodu. Neuvažujte priority transakcí (všechny jsou stejné). Zařízení (číslovaná od 0) jsou obsazována prioritně (nižší číslo zařízení = vyšší priorita). Události vhodně pojmenujte (např. *"P0 obsazuje Z0"*, *"P2 jde do fronty"*).

   Nakreslete časový diagram činnosti simulátoru a napište:

   - Seznam všech událostí provedených v čase $t_1 = 6$.
     Podmínky: každá na zvláštní řádek, události musí být ve správném pořadí (napište PROČ toto pořadí, jinak -3b), a včetně *všech provedených operací plánování do kalendáře* (jinak -4b). Nepište žádné jiné události.
   - Obsah kalendáře v čase $t_2 = 10.5$ (*POZOR: bude-li chybět, -5b*).
   - Nakreslete Petriho síť systému se značením odpovídajícím času $t_2 = 10.5$.

2. (10 bodů) Napište v jazyku C funkci `double mcintegral()` pro výpočet N-násobného určitého integrálu funkce `double f(double x[], unsigned N)` (podle všech $x_i, i \in \{0, ..., N-1\}$) metodou Monte Carlo (integrál musí být N-násobný, jinak 0b) s požadovanou přesností přibližně 1%. Meze integrálu jsou uloženy v globálním poli struktur `Rozsah` s položkami `.min` a `.max`.

   Předpokládejte, že *neznáte rozsah funkčních hodnot* funkce f.

   Máte k dispozici pouze funkci `double Random(void)` vracející pseudonáhodné číslo v rozsahu $\langle 0, 1)$, `f` a *žádnou jinou*.

fZadání zkoušky 2015/2016 - pouze fotky

řešení:
https://docs.google.com/document/d/1pdsRPDE7qi_8Eq6OsOq8-lusj_30JzXM2CpvsbOkUrM/edit?usp=sharing

3. (20 bodů)

a) Napište vzorec pro kongruentní generátor pseudonáhodných čísel. Jaké rozložení generuje? Co je perioda generátoru a jak byste ji zjistili experimentálně?

b) V ISO C implementujte funkci `double Random(void)` generující rovnoměrné rozložení v rozsahu $\langle 0, 1)$.

c) Mějme funkci hustoty pravděpodobnosti $f(x)$ definovánu takto: $f(x) = 3/4$ pro $x \in \langle 1, 2)$ a $f(x) = 1/4$ pro $x \in \langle 2, 3)$, jinak $f(x) = 0$.
Napište obecný vzorec pro výpočet distribuční funkce $F(x)$ z funkce hustoty $f(x)$.
Nakreslete graf průběhu $f(x)$, distribuční funkce $F(x)$ tohoto rozložení (přesně vyznačte min 3 body průběhu této funkce) a průběh inverzní distribuční funkce $F^{-1}(x)$.

d) V ISO C naprogramujte generátor tohoto rozložení `double Gen1()` s využitím metody vylučovací. Nesmíte použít žádné funkce kromě `Random()`.

e) V ISO C naprogramujte generátor tohoto rozložení `double Gen2()` s využitím metody inverzní transformace. Nesmíte použít žádné funkce kromě `Random()`.

4. (15 bodů) V jazyce ISO C napište funkci pro výpočet jednoho kroku numerické metody:

```
RK_step(double t, double state[], unsigned N, double step);
```

které předáte modelový čas `t`, stav všech integrátorů v poli `state`, jejich počet `N` a délku kroku. Funkce nesmí používat globální proměnné. Vzorec definující tuto numerickou metodu je:

$$k_1 = f(t, y(t))$$

$$y(t+h) = y(t) + hk_2$$

Chování modelu je popsáno funkcí

```
Dynamic(double t, double st[], unsigned N, /*out*/ double in[]);
```

ve které se vypočítají vstupy (`in[i]`) všech `N` integrátorů pro dané stavy (`st[i]`) a aktuální modelový čas `t`. Definujte ji konkrétně pro systém zadný rovnicemi:

$$y'' + 6y' - 3z = 0$$

$$7z' + y = 0$$

a nastavte správnou hodnotu konstanty `N`. K rovnici doplňte nenulové počáteční podmínky.

Definujte všechny potřebné proměnné a napište funkci

```
void RunSimulation(double t1, double t2)
```

implementující algoritmus řízení spojité simulace s využitím funkce `RK_step`. Nezapomeňte *"dokročit"* na čas konce simulace `t2`. Řešení, které bude mít max 30 řádků, stručně komentujte.

5. (10 bodů)

Porovnejte jednokrokové a vícekrokové numerické metody pro řešení diferenciálních rovnic z hlediska efektivity výpočtu a to vzhledem k počtu vyhodnocení pravých stran rovnic a k řádu metody.

Klasifikujte typy chyb těchto metod a nakreslete graf ilustrující vliv délky kroku na velikost lokální chyby. Vysvětlete *proč* vznikají uvedené chyby.

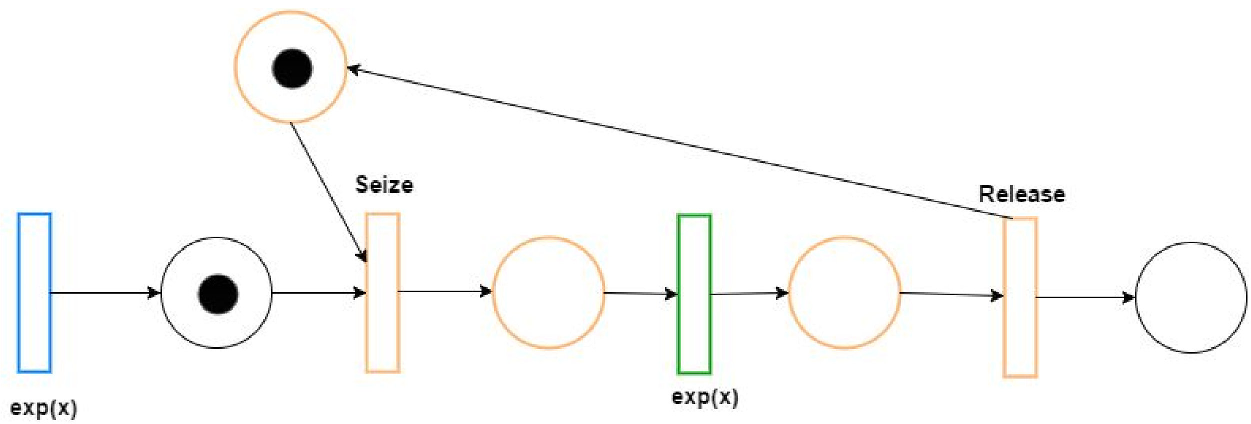
Příprava na půlsemestrální zkoušku

V této části jsou ponechány komentáře a dialogy (většina z nich), které Vám poskytnou ultimátní pohled do procesu řešení petriho sítí a v neposlední řadě taky náhled do duší individuí studující tuto fakultu.

Půlsemestrálka teoretická část

M/M/1 - zakresli pomoci PN a demonstrujte SimLibem.

- M - exponenciápní doba přístupu
- M - exponenciální doba obsluhy
- 1 - jedna obslužná linka
- PN:



- kód z dema:

```
Facility Linka("Obsluzna linka");
Stat dobaObsluhy("Doba obsluhy na lince");
Histogram dobaVSystemu("Celkova doba v systemu", 0, 40, 20);
class Generator : public Event {
    void Behavior() {
        (new Transakce)->Activate();
        Activate(Time + Exponential(11));
    }
};
int main(){
    Init(0, 1000);
```

```
    (new Generator)->Activate();
    Run();
    dobaObsluhy.Output();
    Linka.Output();
    dobaVSystemu.Output();
}
class Transakce : public Process {
    void Behavior() {
        double tvstup = Time;
        double obsluha;
        Seize(Linka);
        obsluha = Exponential(10);
        Wait(obsluha); // Activate(Time+obsluha);
        dobaObsluhy(obsluha);
        Release(Linka);
        dobaVSystemu(Time - tvstup);
    }
};
```

2. Popište strukturu obslužné linky s kapacitou 1 a ve formě pseudokódu zapište operace obsazení a uvolnění linky procesem.

Facility (zařízení)

- **je obsaditelné procesem (výlučný přístup)**
- **obsahuje 2 fronty požadavků:**
    a. **vnější (fronta čekajících požadavků)**
    b. **vnitřní (fronta přerušených požadavků)**

každé zařízení má vlastní frontu -> pole zařízení

linka je buď volná nebo obsazena

```
//obsazeni linky
Facility::Seize(Process *proc) {
        if (In != NULL) {
                Q1.Insert(proc);
                proc->Passivate();
        }
        In = proc;
}

//uvolneni
Facility::Release() {
        In=0;
        if (Q1.Length()>0) {
```

```
        (Q1.GetFirst())->Activate();
    }
}
```

3. Priorita obsluhy u SHO, demonstrujte SimLibem a PN.

4. Uvést pseudokódy seize a release u facility s prioritou obsluhy a uvést, jaké musí mít taková fronta atributy.

- fronta musí také obsahovat informaci o prioritách procesů
- pokus o prepis ze zdrojaku simlibu a zjednoduseni:

```
Facility::Seize(Process *proc, priority p) {
  //pokud neni obsazena tak neresim
  if (!Busy()) {
    in = proc;
  }

  //pokud mame vetsi prioritu tak prerusim aktualni obsluhu
  if (p > in){
     //ulozim procesu kolik casu mu zbyvalo do dokonceni
     in->RemainingTime = in->ActivateTime - Time;
     //pak vlozim do fronty (asi ta kde cekaji prerusene)
     Queue2.insert(*in);
     //uspim a vezmu ten co prerusil;
     in->passivate();
     in = proc;
  }

  //pokud neni dost namachrovany aby prerusil tak jde cekat
  else
  {
     Queue1.insert(*proc, p);
     proc->passivate();
  }
}

Facility::Release() {
//ten kod v simlibu je sileny
//asi bych to zjednodusil proste na: vezmi z hlavni fronty pokud je tam zas
nekdo s mega prioritou a jinak nechej pokracovat nekoho z fronty ktera ma
prerusene procesy
  if (Queue1.first.priority > Queue2.first.priority)
     Queue1.first->activate();
```

```
  else
    Queue2.first->activate();
}
```

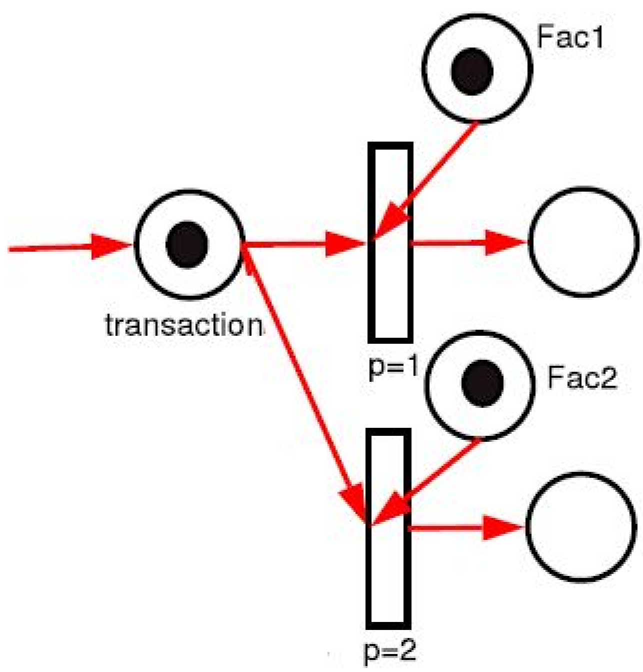
5. Popsat všechny typy priorit používané u SHO a Petriho sítí a každou popsat kódem v SimLibu a PN.

priorita : transakce přistupuje prioritně k jednomu ze zařízení

Priorita procesu

- (atribut třídy Process) – priorita při řazení do front

```
class MProc : Public Process {
...
public:
	MProc() : Process( PRIORITA1) { };
	void Behavior() {
		Priority = 3; //zmena priority
		Seize(F);
		Priority = 0; //implicitni priorita
	}
};
```



Priorita obsluhy:

- výhradně u Seize
- modelování poruch
- proces s vyšší p.o. vyřadí obsluhovaný proces (do vnitřní fronty)
- jsou DVĚ různé fronty

Seize(Fac);
V obsluze je proces A se standardní prioritou obsluhy (0)
…
Seize(Fac,1);
Jiný proces B žádá o obsluhu s vyšší prioritou obsluhy. Proces A je odstaven do vnitřní fronty a do obsluhy se dostává B. Při uvolnění zařízení procesem B se vrátí k rozpracované obsluze proces z vnitřní fronty s nejvyšší prioritou obsluhy a dokončí se jeho obsluha.

6. Popište části, ze kterých se skládají systémy hromadné obsluhy. Napište pseudokód pro operace zabrání a uvolnění obslužné linky transakcí.

- sho typicky obsahuje:
    - transakce (procesy) a popis jejich příchodů
    - obslužné linky (různé typy) a popis obsluhy
    - fronty (různé typy) ve kterých transakce čekají

    tady je kód ze simlibu, nvm na kolik je to požadovaná odpověď (demo 2):

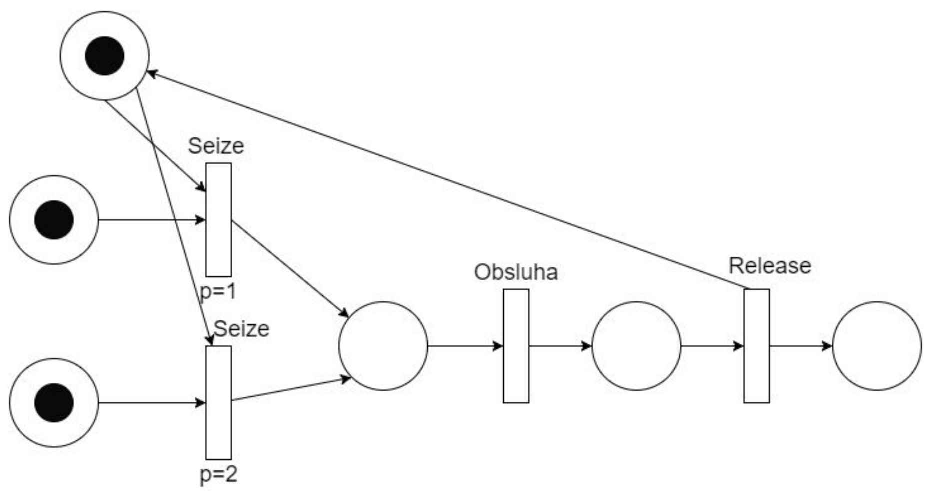
```
//zde jen demonstrace pouziti
Facility Fac;
....
Seize(Fac);
Wait(Exponential(X));
Release(Fac);


Facility::Seize(Process *proc) {
  if (In != NULL) {
    Q1.Insert(proc);
    proc->Passivate();
  }
  else //(v pdfku to neni ale myslim ze by to tu melo byt ne?) - nemělo,
jinak by po tom, co se proces probere a přejímá link nenastavil hodnotu in na
svoji referenci
  In = proc;
}

Facility::Release() {
  In=0;
```

```
  if (Q1.Length()>0) {
    (Q1.GetFirst())->Activate();
  }
}
```

7. Popište varianty modelování priorit transakcí na obslužných linkách a zakreslete je pomocí PN.

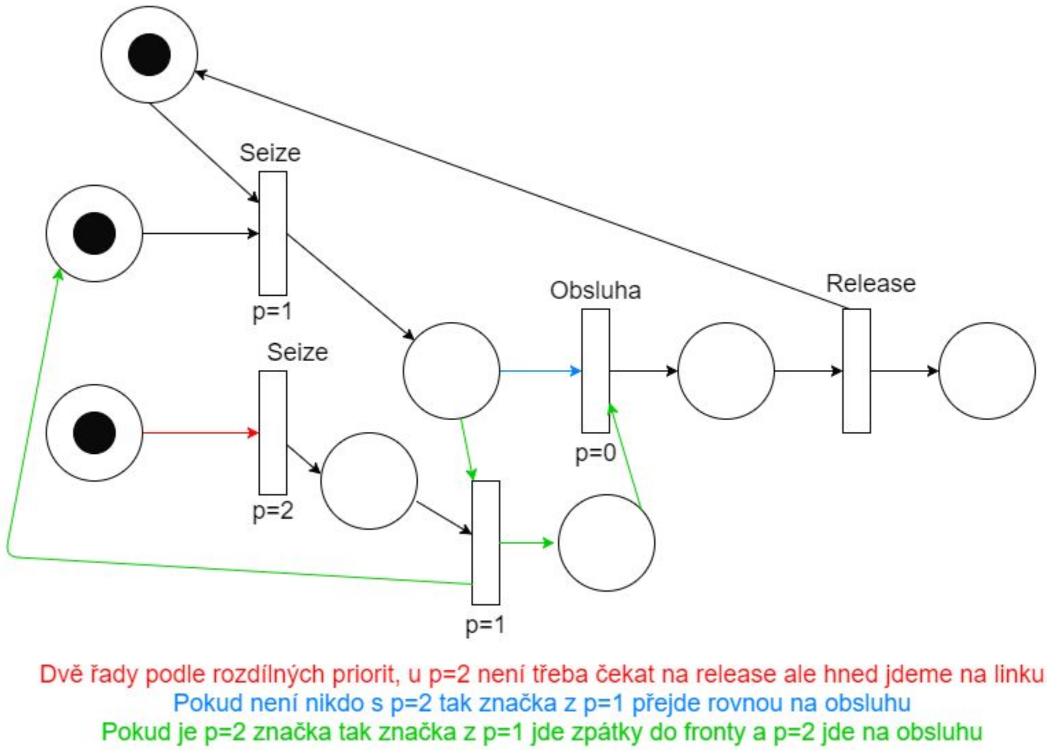
- přicházející požadavky nejsou rovnocenné, může být i více prioritních úrovní, u jedné linky i více front s různými prioritami
- je obsluhován požadavek s nižší prioritou a přijde požadavek s vyšší, mohou nastat 4 možnosti obsluhy:

1. započatá obsluha se normálně ukončí (slabá priorita)

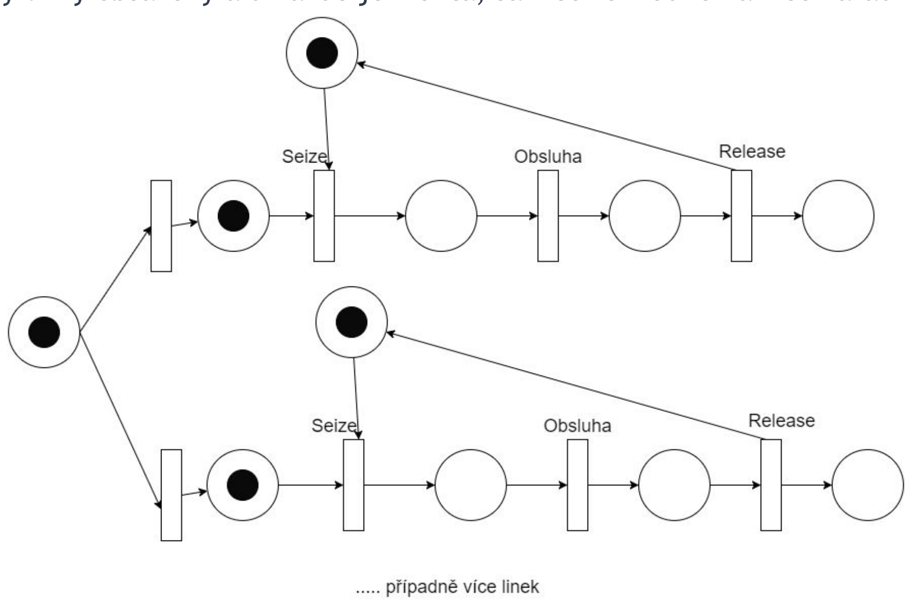


Dvě řady podle rozdílných priorit

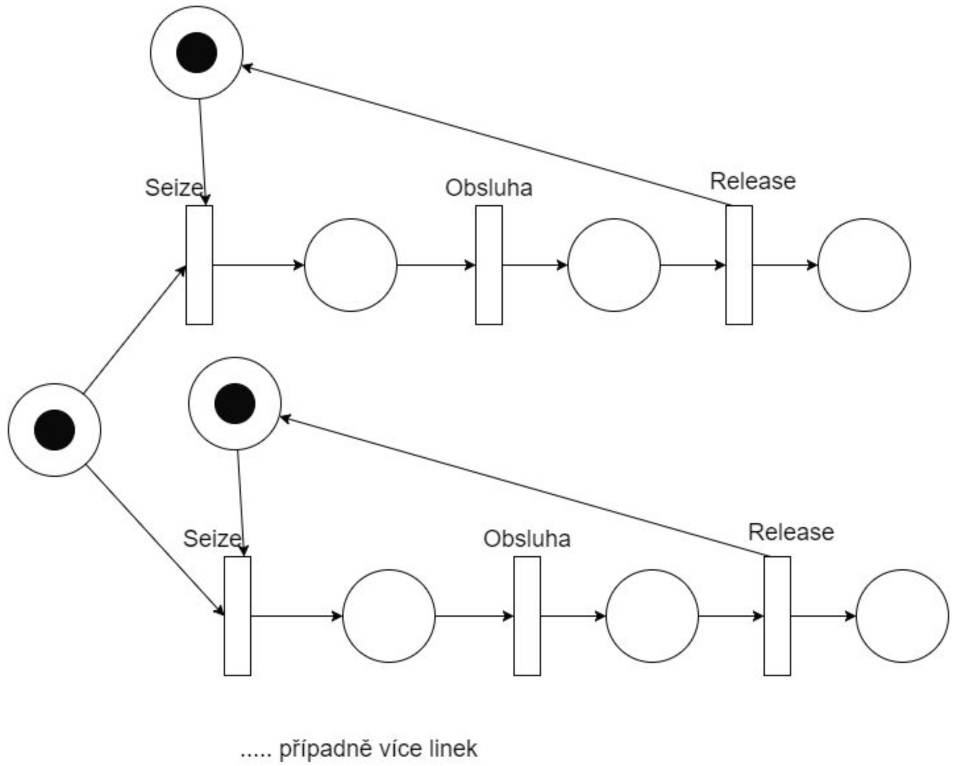
2. obsluha se přeruší a začně obsluha požadavku s vyšší prioritou obsluhy (silná priorita)
   - požadavek jehož obsluha byla přerušena:
     - a) odejde ze systému neobsloužen
     - b) vrací se do fronty
       - * kde je znova obsloužen od začátku
       - * nebo jen naváže a obsluha pokračuje kde byla přerušena



3. jsou-li všechny linky obsazeny a u každé je fronta, sám se rozhodne kam se zařadí



4. je-li jedna společná fronta tak požadavek vstoupí do té linky, která se nejdříve uvolní

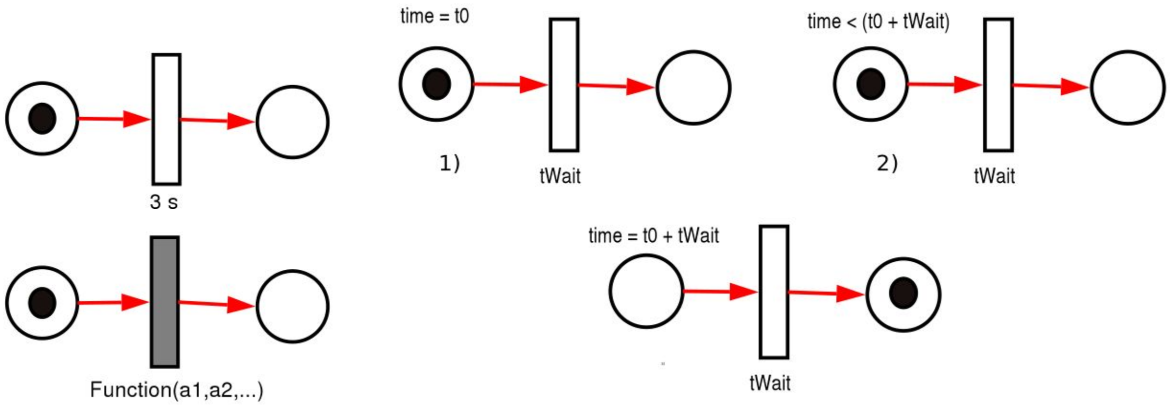


8. Model, simulace, validace, verifikace.

- model - napodobenina systému jiným systémem (reprezentace znalostí)
  musí napodobovat všechny, pro nás podstatné vlastnosti původního systému
- simulace - získávání nových znalostí o systému experimentováním s jeho modelem
- validace - ověřování platnosti modelu
  cílem je dokázat, že model skutečně odpovídá původnímu systému
  nelze dokázat absolutně, je to pouze míra použirtelnosti výsledků
- verifikace - ověření korespondence abstraktního a simulačního modelu (izomorfní vztah mezi AM a SM)

9. Modelování, modelový čas, časovaný přechod.

- modelování - proces vytváření modelů systému na základě našich znalostí
- modelový čas - časová osa modelu, modeluje reálný čas z původního systému
  při simulaci nemusí být synchronní s časem reálným
- časovaný přechod je prvek Petriho sítí
  jedná se o přechod s paramez*ní* (může být konstatní, nebo náhodně generovaná [obrázek vlevo])
  sémantika: [obrázek vpravo]
  1. pokud je přechod v čase t proveditelný, spustí se odpočet
  2. po celou dobu odpočítávání se nemění stav značek
  3. na konci doby se provede přemístění značek



10. Definujte strukturu diskrétního simulátoru se zarážkou. Reálný čas, simulační čas, modelový čas.

- zarážku tvoří koncový čas v kalendáři, položky kalendáře za tímto časem se již neaktivují

- reálný čas - čas ve kterém probíhá skutečný děj v reálném systému
- modelový čas - časová osa modelu, modeluje reálný čas z původního systému
  při simulaci nemusí být synchronní s časem reálným
- strojový čas - je čas CPU spotřebovaný na výpočet programu (závisí na složitosti programu, počtu procesorů atd., nesouvisí přímo s modelovým časem)

11. "V pseudokódu definovat třídu pro obslužnou linku a v ní potřebné struktury a operace pro obsazení a uvolnění linky"

diskr2-2011.pdf slide9, slide10

Půlsemestrálka praktická část - Petriho sítě

Příklad ústředna (možná jiné konstanty)

Firma F používá telefonní ústřednu s kapacitou 20 současných telefonních hovorů. Zákazníci volají různým zaměstnancům firmy průměrně 30krát za hodinu (Poissonův proces příchodů).

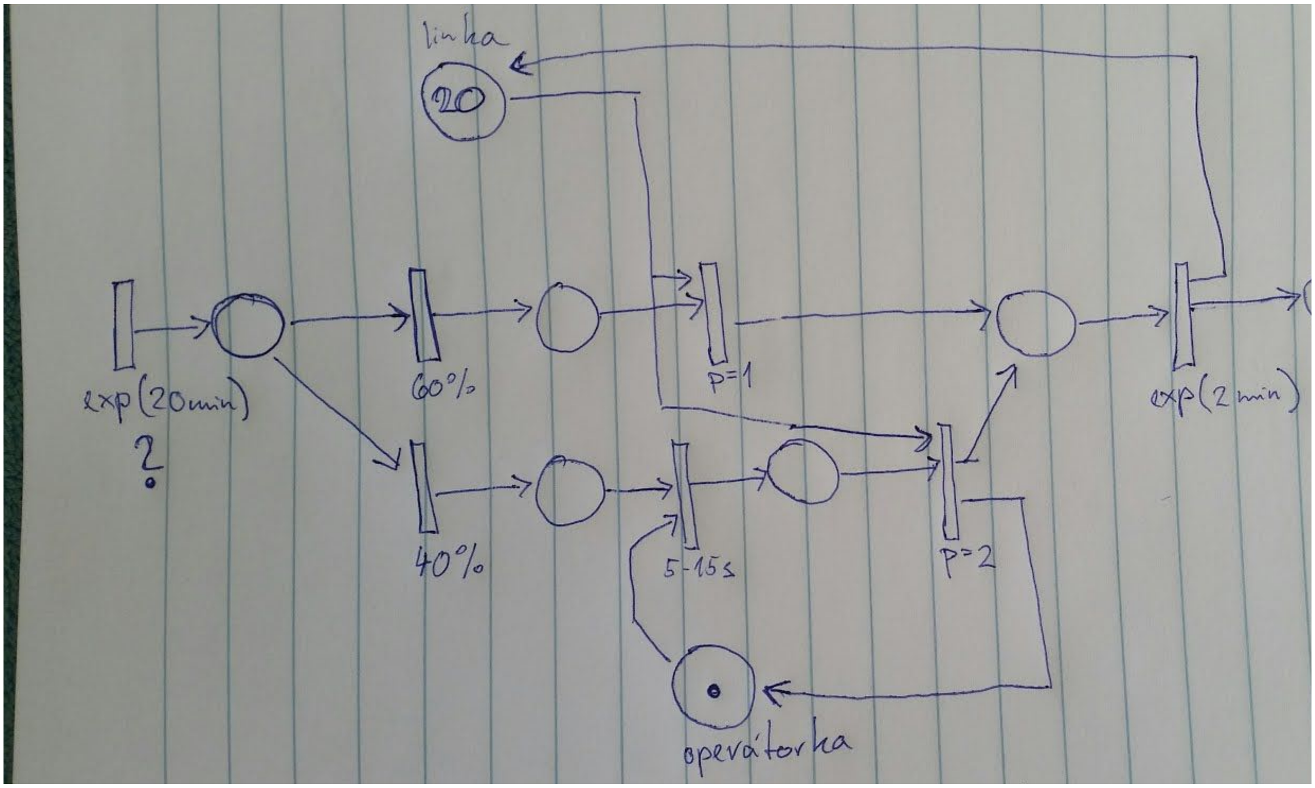
Existují dvě možnosti: 60 procent telefonujících zná místní telefonní číslo zaměstnance (klapku) a ústředna mu přidělí volnou linku automaticky.

Zbylých 40 procent volajících klapku nezná a jsou přepojeni na operátorku.

Zjištění klapky u~operátorky trvá dobu danou rovnoměrným rozložením od 5 do 15 sekund. Poté má volající spojovaný přes operátorku vyšší prioritu při obsazování některé z linek.

Doba hovoru se řídí exponenciálním rozložením se střední hodnotou 2 minuty.

Po skončení hovoru a uvolnění linky volající požadavek opouští systém.



Neni to spíš takhle, protože by mel cekat na volnou operátorku?
https://ctrlv.cz/shots/2015/11/08/xKv3.png,
Souhlasím 2x

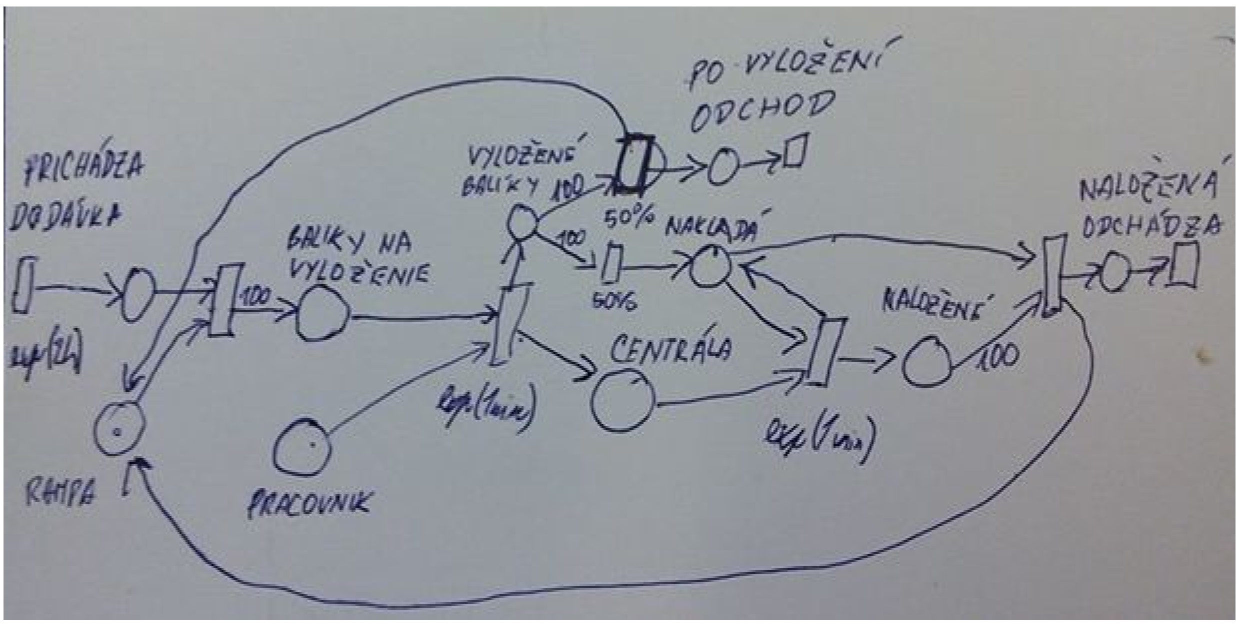
Příklad se zvukovou kartou

Mějme jednoprocesorový počítač se zvukovou kartou, na kterém běží 10 procesů. Jeden z nich je speciální a zapisuje do zvukové karty vzorky, které je schopen produkovat rychlostí 500 tisíc za sekundu. Karta má vyrovnávací paměť o velikosti 10 tisíc vzorků a je nastavena tak, že přehraje 100 tisíc vzorků za sekundu. Pokud se vyrovnávací paměť dostatečně vyprázdní (a je volný procesor), je procesor prioritně přidělen zvukovému procesu, který ji začne plnit po blocích 100 vzorků dokud nebude zcela plná a potom se vzdá procesoru. (Doporučení: tečka = volné místo pro jeden blok 100 vzorků)

Všechny ostatní procesy mají stejnou prioritu a je jim spravedlivě (round-robin) přidělován procesor s časovým kvantem 10ms. Procesy se mohou vzdát procesoru s pravděpodobností 20 procent v libovolném okamžiku jim přiděleného časového kvanta. Dobu potřebnou pro přepnutí kontextu zanedbáváme.

1 Logisticka centrala

Do logistické centrály přijíždí dodávky v intervalech daných exponenciálním rozložením se středem 2 hodiny. Dodávka přistupuje k centrále přes jednu nakládací/vykládací rampu. Dodávka přiveze 100 balíků ke zpracování. Dobu vyložení balíku zanedbáváme. Zpracováním přivezených balíků se zabývá jeden pracovník. Doba zpracování každého jednotlivého balíku se řídí exponenciálním rozložením se středem 1 minuta. Zpracované balíky jsou umístěny v centrále. Kapacitu centrály pro uložení přivezených nebo zpracovaných balíků nezkoumáme. Polovina dodávek po vyložení balíků nakládá zpracované balíky až po svou kapacitu 100 balíků, druhá polovina po vyložení okamžitě opouští systém. Doba naložení každého jednotlivého balíku se řídí exponenciálním rozložením se středem 1 minuta. Po plném naložení dodávka odjíždí a další její chování nezkoumáme. Poznámka: Pracovník zpracovávající přivezené balíky je nezávislý proces.



Ten pracovník se nevrací? Navíc kde je stav (Seize) pro zabrání linky Pracovník? Myslím že je chyba když tento stav chybí a je tam hned ten časovaný. (v tomto případě to asi je jedno ale všude to tak dělají)

1x pracovnika treba vratit

Neměli by tam být odkonce 2 pracovníci, zvlášť na zpracování a nakládání?

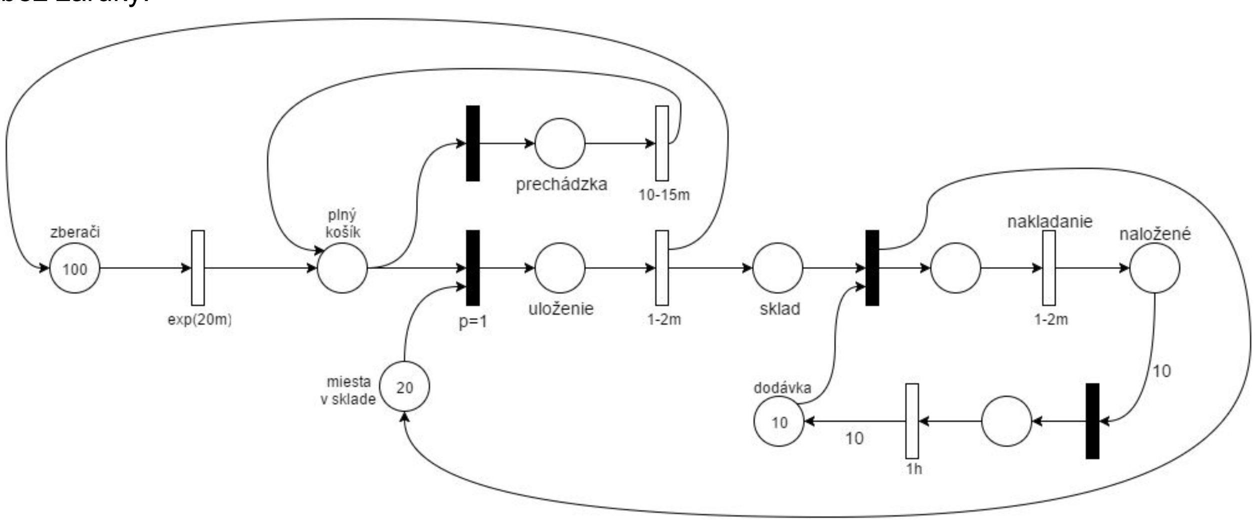
- mozno jeden pracovnik na obe cinnosti?
- obojí by dávalo smysl :D
- to si naloží řidič ne? :D
- :D :D :D :D
- ale tak jo :D v zadani je ze pracovnik se zabyva zpracovanim, nic vic

ako sa sem vklada obrazok?

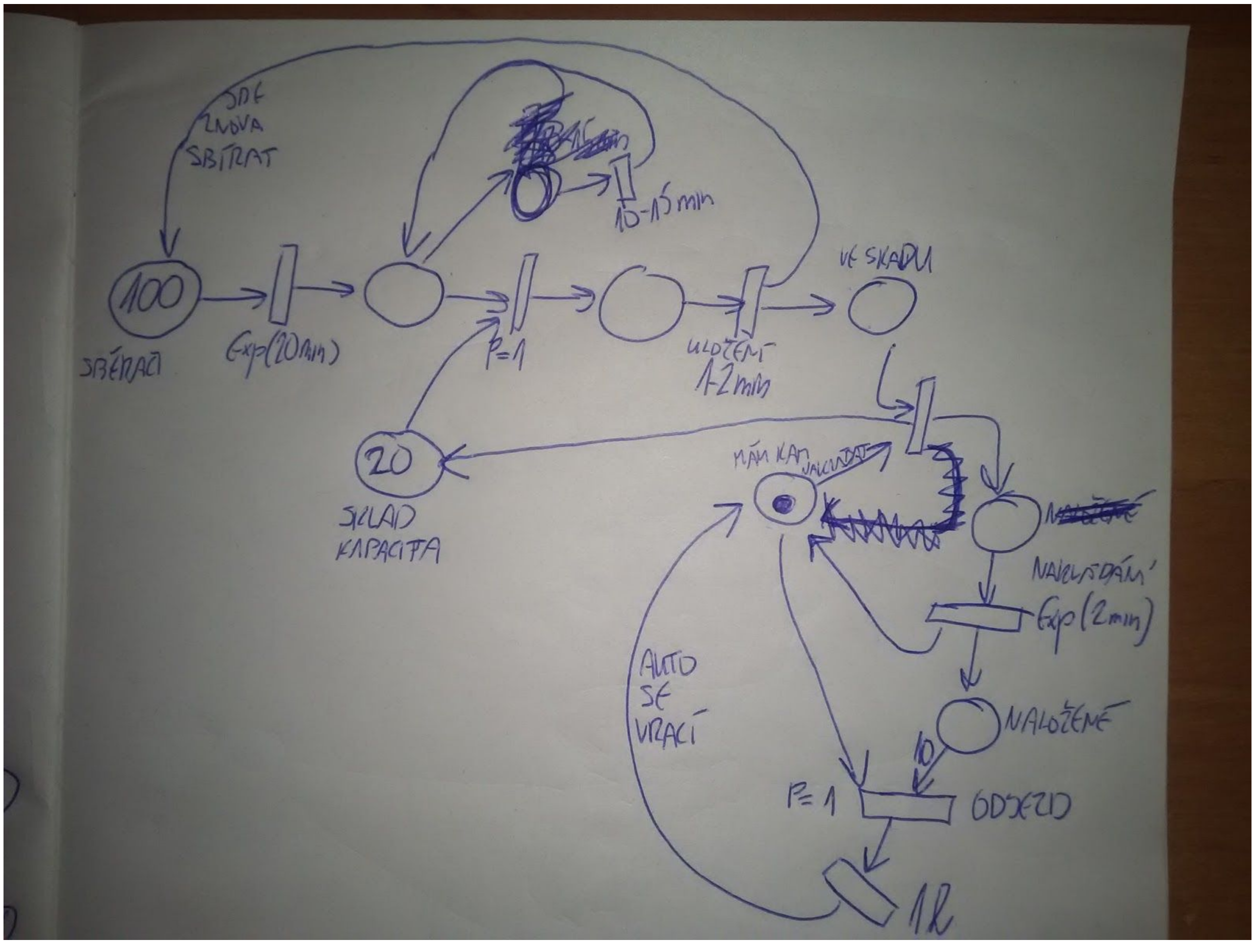
2 Chmel

Na chmelové brigáde je 100 sberacu· chmele. Doba sberu chmele do kosíku se rídí exponenciálním rozlozením se stredem 20 minut. Po naplnení kosíku jde sberac kosík ulozit do skladu s kapacitou 20 kosíku. Pokud ve skladu není aktuálne volné místo pro ulození kosíku, sberac se vydá s kosíkem
na 10-15 minut na procházku. Po návratu opet zkoumá volnou kapacitu skladu (pocet procházek
je neomezený). V prípade volné kapacity skladu zabere ulození kosíku do skladu 1-2 minuty rovnomerne. Po ulození plného kosíku s nasbíraným chmelem obdrzí sberac volný kosík a vrací se ke sberu. Pocty kosíku v systému nesledujeme a zanedbáváme i doby presunu sberace mezi polem
a skladistem (tzn. není tam sklad volných kosíku). V systému pracuje jedna prepravní dodávka s kapacitou 10 kosík·. Nalození kazdého jednotlivého kosíku do dodávky trvá dobu, která se rídí exponenciálním rozlozením se stredem 2 minuty. Okamzikem zapocetí nakládání kosíku dodávkou se uvolnuje jedna pozice ve skladu. Po plném nalození dodávka odjízdí a vrací se prázdná za jednu hodinu.
Poznámka: Kapacitu skladu plných kosík· lze modelovat místem v Petriho síti se zadanou kapacitou.

bez záruky:



- použitie kapacity v stave “sklad” by to značne zjednodušilo, ale Hrubý tuším povedal, že tomu sa máme vyhýbať
- nesouhlasím s obr., začaly by se nakládat všechny košíky naráz
    - jn … to by snáď vyriešila kapacita = 1 toho stavu pred prechodom “nakladanie”
- nema byt u toho nakladani ve stavech vyjadrena kapacita?



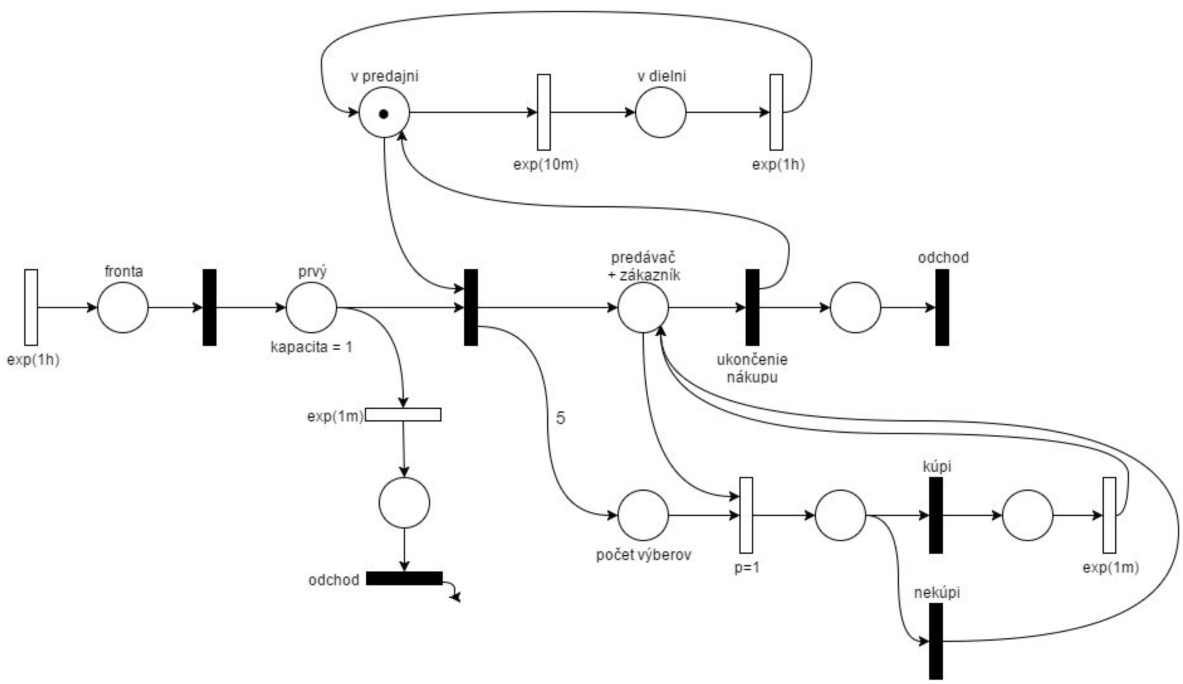
tohle reseni je spatne, nesmi byt: prechod -> prechod, stav -> stav
Pravda, patří mezi to kolečko

3 Prazirna kavy

Do prazírny kávy píchází zákazníci v intervalech daných exponenciálním rozlozením se stredem 1 hodina. V prazírne pracuje jeden prodavac. Prodavac se premistuje mezi prodejnou a dílnou na prazení kávy. Zákazníci se radí do fronty a cekají na prodavace s tím speci_kem, ze vzdy první ve fronte po minute cekání opoustí prodejnu neobslouzen (tzn. netrpelivý je pouze první zákazník ve fronte). Pokud je prodavac v prodejne, obsluhuje zákazníky do vycerpání fronty. Pokud je prodavac v prodejne, pak po 10 minutách cekání na zákazníka odchází do dílny, kde prazí kávu. Doba prazení se rídí exponenciálním rozlozením se stredem 1 hodina. Po dokoncení práce se prodavac vrací do prodejny (doby presunu zanedbáváme). Zákazník si

kupuje 0 az 5 balícku kávy. Výber a zakoupení kazdého jednotlivého balícku mu zabere dobu, která se rídí exponenciálním rozlozením se stredem 1 minuta.
Poznámka: Zákazník tedy provede práve 5 iterací nákupu, kde kazdý z nich je pravdepodobnostní.

Zase bez záruky:



S tímto asi neshouhlasím - tady zákazník může neustále iterovat mezi stavem predavač + zákazník a přechodem s prioritou p=1, pak si kúpí/nekúpí a jde zase do stavu, ze kterého ma prioritní přechod…. A nic ho nedonutí k ukončení nákupu, ne?
Add: Omezení kapacity pro prvního zákazníka je špatně. Protože jakmile si tento zákazník zabere prodavače, tak je stále na prvním místě fronty. V tomhle řešení se ale první místo uvolní a přijde na něj další zákazník, kterému se počítá timeout, ikdyž tam prodavač je.
Nebude iterovat neustále, pouze 5x, pak nebude přechod proveditelný, protože nebudou značky v místě “počet výběrů”
S timeoutem pro prvního asi souhlasím

Půlsemestrálka zadání 2013/2014

Varianta A

1.

Definujte datovou strukturu pro diskretni simulator a formou pseudokodu implementujte algoritmus diskretniho simulatoru se zarazkou modeloveho casu.

Skupina A:

Definujte strukturu diskretniho simulatoru se zarazkou + napsat pseudokod.

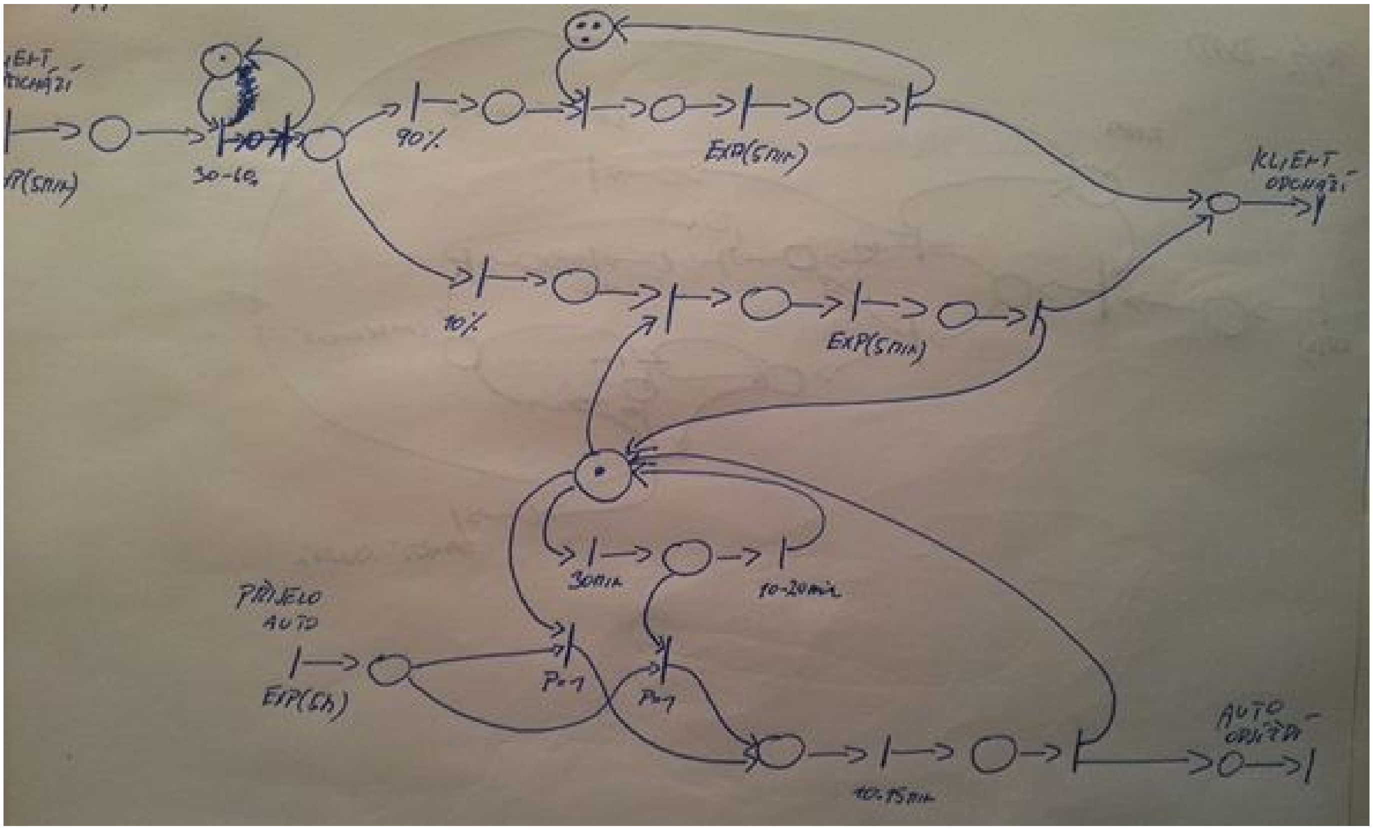
Kód:

```
while (!kalendar.empty())
{
    Event ev = kalendar.pop();
    if (ev.time > T_END)
        break;
    Time = ev.time;
    ev.cinnost();
}
```

2.

Do bankovni pobocky prichazeji klienti v intervalech dany exponencialnim rozdelenim se stredem 5 minut. V pobocce pracuji 3 pracovnici na prepazce a 1 vedouci pobocky. Klient si po prichodu do pobocky vezme listek ze stroje pro evidenci poradi, pouziti stroje zabere 30-60 sekund rovnomerne. 10% klientu pozaduje specialni poradenstvi, ktere vykonava vedouci, ostatni klienti jsou obsluhovani na prepazkach. Obsluha na prepazce i poradenstvi trvaji exp(5) minut. Vedouci je pocatecnim stavu na sve prepazce. Pokud je k jeho prepazce fronta prazdna 30 minut vydava se na kontrolu pobocky(10-20 minut). V intervalech exp(5) h prijizdi auto, ktere ma pro vedouciho prioritu. Pokud neobsluhuje klienta, zapocne obsluhu auta, ktera zabere 10-15 minut. Po obsluze auta se vraci k prepazce.

Není tam 3x zbytečné místo s přechodem? U obsluhy klientu za Exp(5min) a u auta za přechodem 10-15min? Souhlasím že ty místa za přechody s exp(5min) jsou navíc.
Jak je ten stroj na lístky tak nemusí tam být ještě před tím stav seize? jinak souhlas.
já to mám stejně, ty místa tam být musí, simuluje to odchod zákazníka/auta ze scény.



Zadání B:

1. Otázka - teorie (4 b)

M/M/1 - zakreslit pomocí PN a zapsat kód pro SimLib.

PN viz výše

```
Facility Linka("Obsluzna linka");

class Generator : public Event {
      void Behavior() {
            (new Transakce)->Activate();
            Activate(Time + Exponential(11));
      }
};

int main() {
      Init(0, 1000);
      (new Generator)->Activate();
      Run();
```

```
}

class Transakce : public Process {
      void Behavior() {
            double obsluha;

            Seize(Linka);
            obsluha = Exponential(10);
            Wait(obsluha); // Activate(Time+obsluha);

            Release(Linka);
      }
};
```

2. Otázka -
Petriho síť (Je to z hlavy, tak mě kdyžtak opravte)
Systém, který zpracovává hrášek - má tři fáze - zpracování hrášku, tepelná úprava a balení.
S časem Exp(3h) přijíždí dodávka s natrhaným hráškem. Všechen hrášek z dodávky se nasype do stroje na zpracování (v systému jen jeden) a ten ho za 30-50 minut zpracuje, jako výsledek pak vyprodukuje 100ks polotovarů, které dále v systému putují jednotlivě. Linka pro tepelné zpracování má kapacitu 50, tepelné zpracování zabere 1 hodinu. Po tepelném zpracování následuje balení. Linka pro balení má kapacitu 1 a balení trvá 1 minutu.
Každých Exp(10h) dojde k poruše na balícím zařízení. Tato porucha bude opravena po 30-80 minutách. K poruše dojde v průběhu balení. Právě zpracovávaný hrášek je znehodnocen. Po dobu poruchy přestane přijímat polotovary linka pro tepelné zpracování, ale dokončí již započaté zpracování. Hrášek po tepelné úpravě před zabalením má dobu trvanlivosti 1h. Po této době je znehodnocen. V systému udržujte místo s počtem znehodnoceného hrášku.

Skupina C:

1. otázka:

Popište strukturu obslužné linky s kapacitou 1 a ve formě pseudokódu zapište operace obsazení a uvolnění linky procesem.

2.příklad (Petriho síť):

Server má 1 procesor a 1 síťovou linku a podporuje multitasking. Síťová linka může v jeden okamžik buď přijímat zprávy a ukládat je do nekonečného vstupního bufferu, nebo z nekonečného výstupního bufferu data vysílat. Ze sítě přicházejí požadavky na databázovou službu exp(1s). Na serveru běží 10 uživatelských úloh a jedna speciální - databázová úloha. Operační systém přiděluje procesor běžně tak, že v 10% případů ho dostane databázová úloha, v 90% uživatelská. Uživatelská úloha se vykonává po 10ms. Databázová úloha postupně zpracuje všechny požadavky ze vstupního bufferu, zpracování každého trvá exp(1ms). Zpracované požadavky se posílají do výstupního bufferu. Pokud je ve vstupním bufferu více jak 100 požadavků, procesor je okamžitě přidělen databázové úloze - je sebrán prováděné uživatelské.

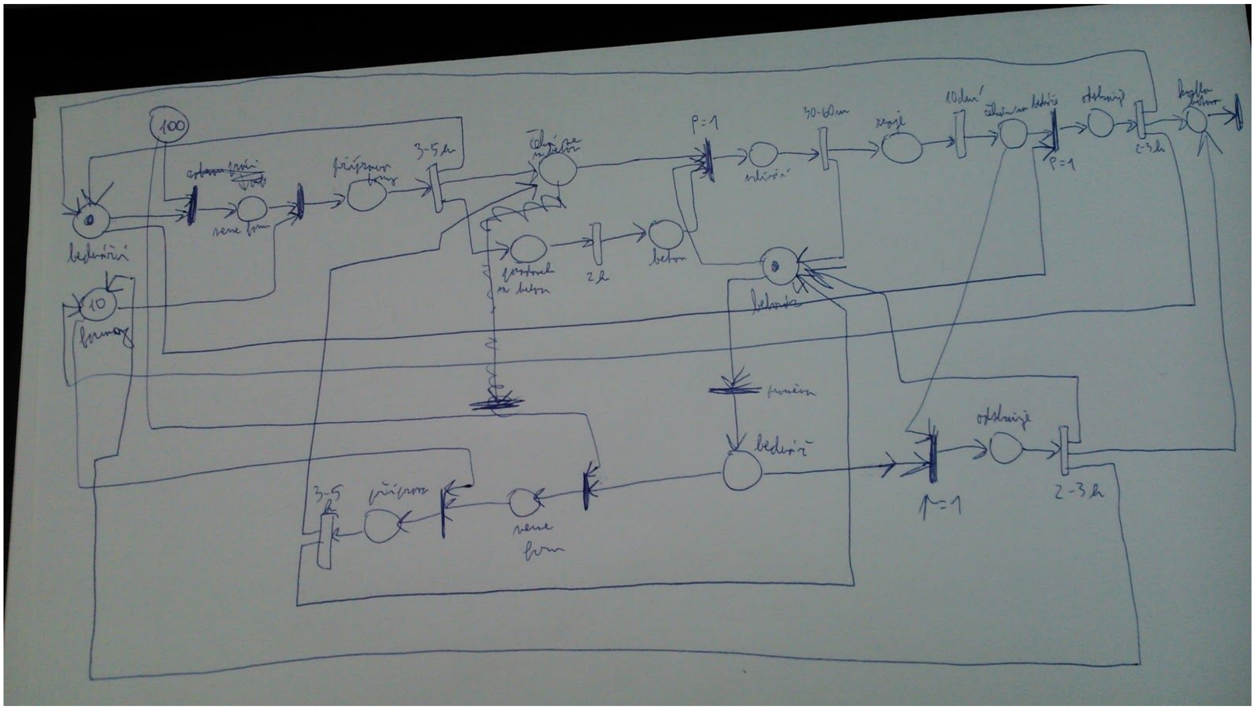
Zadání D:

1. Otázka (4 b)

Definujte pojmy model, modelování, modelový čas, simulace, časovaný přechod u stochastické Petriho sítě.

2. Příklad (6b)

Pro dokončení stavby je potřeba vykonat 100 elementárních prací. Na stavbě pracuje jedna parta bednářů a jedna parta betonářů. Pro dokončení jedné elementární práce je potřeba vykonat tuto posloupnost: 1) sestavit bednění licí formy, trvá 3-5 hodin (dělají bednáři), 2) nalít beton do formy, trvá 30-60 minut (dělají betonáři), 3) nechat beton uzrát, trvá 10 dní, 4) odstranit bednění licí formy, trvá 2-3 hodiny (dělají bednáři). Na stavbě je 10 licích forem. Pokud je někde potřeba bednění sestavit a na jiném místě odstranit, bednáři prioritně bednění odstraňují.
Po dokončení sestavení licí formy se odešle požadavek na dodávku betonu, která dorazí za 2 hodiny od zadání požadavku. Pokud betonáři nemají dodávku betonu, přidávají se na jeden úkol k bednářům.



Může to být takhle? Nevím jestli jsem správně pochopil tu dodávku betonu po 2h a těch 10 forem.

Půlsemestrálka zadání 2012/2013

skupina A

1)Kalendář se zarážkou

skupina B

1)Teoreticka otazka (4b) bola z minuleho roka: Definujte a pomoci pseudokodu popiste diskretni simulator s casovou zarazkou. Definujte potrebne datove typy. Popiste co je to realny cas, simulacni cas a modelovy cas.
2) textilka/svadleny:

skupina C

1) Model M/M/1, demonstrujte SIMLIBem
2) Automat na kavu

skupina D

1) priorita obsluhy u SHO, demonstrujte SIMLIBem a PS
2) spalovna odpadu

Půlsemestrálka zadání 2011/2012

skupina A

1) 4b - Definujte a pomoci pseudokodu popiste diskretni simulator. Definujte potrebne datove typy. Popiste co je to realny cas, simulacni cas a modelovy cas.
2)řešení
https://fituska.eu/download/file.php?id=8876&mode=view

skupina B

zadání
https://fituska.eu/download/file.php?id=8877&mode=view
řešení
https://fituska.eu/download/file.php?id=8889&mode=view

skupina C

skupina D

1)
2) řešení
https://fituska.eu/download/file.php?id=8875&mode=view

Nekoukat.
...když už ses kouknul, je to aspoň dobře?? :(@) << jazykové tornádo

t=3
uvolnění zařízení z0
naplánování obsazení zařízení z0 procesem p2 z fronty
vygenerování procesu p3 a naplánování obsazení
obsazení zařízení z0 procesem p2
naplánování uvolnění zařízení z0 v čase t+3
obsazení zařízení procesem p3 (obě jsou vytížená => proces p3 jde do fronty)
naplánování vygenerování procesu v čase t+1

dddd