IMS Democvičení #2

Simulační experimenty
SIMLIB/C++

Další akce...

- 2. 11. - Cvičení z modelování Petriho sítěmi.
- 9. 11. - Seminář o řešení projektů.
- 16. 11. - Půlsemestrální písemka.

Cíl počítačových simulací

- Vycházíme z (simulačního) modelu B systému A
- (Model systému A je jiný systém B, který mu je určitým způsobem podobný)
- Cílem modelování je vytvořit napodobeninu nějakého systému (reality)
- Simulace = experimentování s modelem
- *Cílem simulace modelu B je získat nové informace o systému A*

Forma experimentu

- Počítačový simulační model je obvykle ve formě nějakého PROGRAMU
- (V našem případě je to program v jazyce C++ s použitím knihovny SIMLIB)
- Model v SIMLIB je program jako každý jiný!!!! (jako každý jiný C++ program)
  - (otázka “jak se v SIMLIBu udělá cyklus nebo vynásobí dvě čísla” by se neměla objevovat!!!)

Forma experimentu

- Simulační experiment je spuštění programu se zadáním:
  - struktury modelu (např. obslužná síť)
  - parametrů modelu (např. kapacity, intervaly)
  - parametrů prostředí
- Získávání znalostí
  - sledování chování systému
  - “ladící” výpisy
  - statistiky
- Model je určen k experimentování!

SHO - statistiky

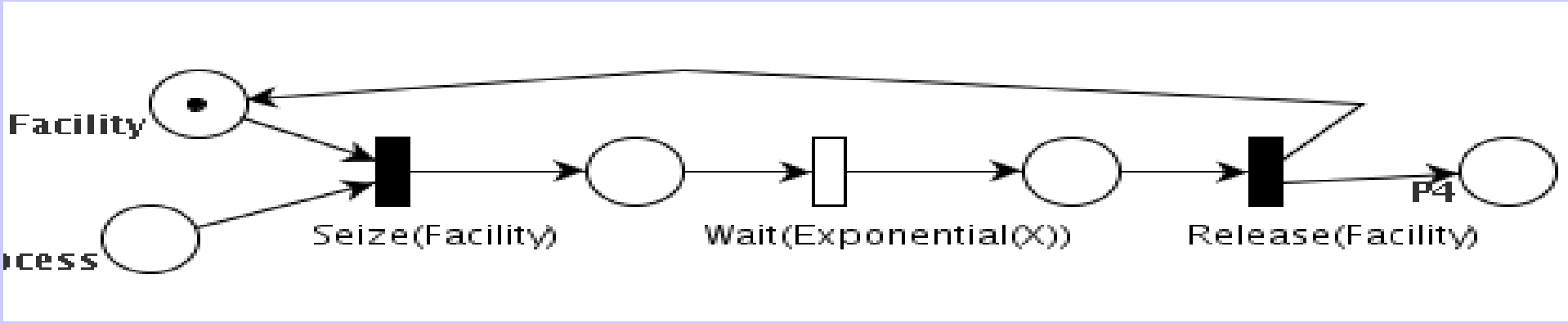
- Transakce, obslužné linky, fronty – časy, využití
- Transakce:
  - celková doba v systému
  - doba čekání v jednotlivých frontách
  - další .... (např. pravděpodobnosti)
- Linky:
  - Celková vytíženost (v %, v čase)
  - Počet přístupů
  - Doba obsluhy (stat – min, max, průměr, rozptyl)

SHO - statistiky

- Fronty:
  - počet transakcí, které vstoupily do fronty
  - počet vystoupivších
  - doba strávená ve frontě - Stat

Procesy v SIMLIB

- Event – události, nepřerušitelné akce
- Process – přerušitelné posloupnosti událostí
- Simulátory založené na událostech/procesech
- Kalendář:
  - Akce (Event, Process) je objekt v systému
  - Aktivuje se s udáním času – vloží se do kalendáře (fronta akcí s časovým atributem)
  - Akce je spuštěna (metoda Behavior)
  - Process je přerušitelný (Passivate) při
    - vstoupení do fronty (Seize, Enter)
    - čekání (Wait)
    - při vlastní žádosti (Passivate)




```
Facility Fac;
....
Seize(Fac);
Wait(Exponential(X));
Release(Fac);
```

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

Facility::Release() {
    In=0;
    if (Q1.Length()>0) {
          (Q1.GetFirst())->Activate();
    }
}
```

Zařízení versus Sklad

- Facility, Store
- Sklad může být chápán jako několik zařízení. Rozdíl je v organizaci fronty.
- N zařízení má N samostatných front
- Sklad M prostředků má JEDNU frontu
- Příklad:
  - 5 pokladen v samoobsluze je 5 (stejných) zařízení
  - přístřešek s vozíky je sklad

```
#define KOLIK 10
Facility Fac[KOLIK]; Queue Q;
class Trans : public Process {
        void Behavior() {
                int kt = -1;
                zpet:
                for (int a=0; a<KOLIK; a++)
                        if (!Fac[a].Busy()) { kt=a; break; }

                if (kt == -1) {
                        Into(Q);
                        Passivate();
                        goto zpet;
                }

                Seize(Fac[kt]);
                Wait(Exponential(30));
                Release(Fac[kt]);
                if (Q.Length()>0) {
                        (Q.GetFirst())->Activate();
                }
        } };
```

Priority

- 2 typy:
  - Priorita procesu (atribut třídy Process) – priorita při řazení do front
  - Priorita obsluhy – výhradně u Seize
- Priorita obsluhy:
  - modelování poruch
  - proces s vyšší p.o. vyřadí obsluhovaný proces (do vnitřní fronty)
  - jsou DVĚ různé fronty
- Chybné použití priority – projekt “WC”

Příklad: M/M/1

- Jedna linka – obsluha exp(10)
- Příchody transakcí – exp(11)
- Využití linky, doby čekání, ...

```
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);
    }
};
```

!!! Počet přístupů, časový rámec

```
+-----------------------------------------------------+
| STATISTIC Doba obsluhy na lince                     |
+-----------------------------------------------------+
|  Min = 0.0859269              Max = 37.7691         |
|  Number of records = 82                             |
|  Average value = 8.64161                            |
+-----------------------------------------------------+
+-----------------------------------------------------+
| FACILITY Obsluzna linka                             |
+-----------------------------------------------------+
|  Status = not BUSY                                  |
|  Time interval = 0 - 1000                           |
|  Number of requests = 82                            |
|  Average utilization = 0.708612                     |
+-----------------------------------------------------+
```

Input queue 'Obsluzna linka.Q1'

```
+-----------------------------------------------------+
| QUEUE Q1                                            |
+-----------------------------------------------------+
|  Time interval = 0 - 1000                           |
|  Incoming  54                                       |
|  Outcoming  54                                      |
|  Maximal length = 5                                 |
|  Average length = 0.90493                           |
|  Minimal time = 0.0344599                           |
|  Maximal time = 36.752                              |
|  Average time = 16.758                              |
+-----------------------------------------------------+
```

Prodloužíme dobu experimentu

```
+-----------------------------------------------------+
| STATISTIC Doba obsluhy na lince                     |
+-----------------------------------------------------+
|  Min = 0.000468028          Max = 83.5522           |
|  Number of records = 9123                           |
|  Average value = 9.82663                            |
|  Standard deviation = 13.7761                       |
+-----------------------------------------------------+
+-----------------------------------------------------+
| FACILITY Obsluzna linka                             |
+-----------------------------------------------------+
|  Status = BUSY                                      |
|  Time interval = 0 - 100000                         |
|  Number of requests = 9124                          |
|  Average utilization = 0.89652                      |
+-----------------------------------------------------+
```

Input queue 'Obsluzna linka.Q1'

```
+-----------------------------------------------------+
| QUEUE Q1                                            |
+-----------------------------------------------------+
|  Time interval = 0 - 100000                         |
|  Incoming  8200                                     |
|  Outcoming  8184                                    |
|  Maximal length = 41                                |
|  Average length = 7.82557                           |
|  Minimal time = 0.0207657                           |
|  Maximal time = 469.937                             |
|  Average time = 95.4485                             |
|  Standard deviation = 128.638                       |
+-----------------------------------------------------+
```

Proč není linka 100% vytížená?

```
int bezCekani = 0;

v procesu:
if (!Linka.Busy()) bezCekani++;

v main:
Print("Bez cekani: %d\n", bezCekani);

výpis:
Bez cekani: 940
```

Histogram

- Graf četnosti
- Má interval použití, krok
- Neměl by mít více než 20 položek (statistická vypovídací schopnost)
-
- Histogram celkové doby v systému.....
- (následující histogram ukazuje, jak se to NEMÁ dělat)

HISTOGRAM Celkova doba v systemu

STATISTIC Celkova doba v systemu
Min = 0.0859269 Max = 58.088
Number of records = 82
Average value = 19.6773

from	to	n	rel	sum
0.000	40.000	73	0.890244	0.890244
40.000	80.000	9	0.109756	1.000000
80.000	120.000	0	0.000000	1.000000
120.000	160.000	0	0.000000	1.000000
160.000	200.000	0	0.000000	1.000000
200.000	240.000	0	0.000000	1.000000
240.000	280.000	0	0.000000	1.000000
280.000	320.000	0	0.000000	1.000000
320.000	360.000	0	0.000000	1.000000
360.000	400.000	0	0.000000	1.000000

Min = 0.0012543 Max = 477.479
Number of records = 9123

0.000	40.000	3078	0.337389	0.337389
40.000	80.000	1983	0.217363	0.554752
80.000	120.000	1338	0.146662	0.701414
120.000	160.000	834	0.091417	0.792831
160.000	200.000	693	0.075962	0.868793
200.000	240.000	476	0.052176	0.920969
240.000	280.000	270	0.029596	0.950565
280.000	320.000	208	0.022800	0.973364
320.000	360.000	138	0.015127	0.988491
360.000	400.000	68	0.007454	0.995944
400.000	440.000	27	0.002960	0.998904
440.000	480.000	10	0.001096	1.000000
480.000	520.000	0	0.000000	1.000000
520.000	560.000	0	0.000000	1.000000

Shrnutí dosavadních poznatků

- Simulace je numerická metoda s určitou přesností (vztaženo k analytickému řešení)
  - přesnost lze ovlivňovat (zvyšovat) počtem iterací v simulačním experimentu
  - (počet transakcí, časový rámec simulace)
- Statistiky
  - Vytvářet s cílem podávat statisticky hodnotnou informaci (např. histogramy)
  - Lze tak i najít chyby v modelu nebo systému (podezřelé fronty, ...)

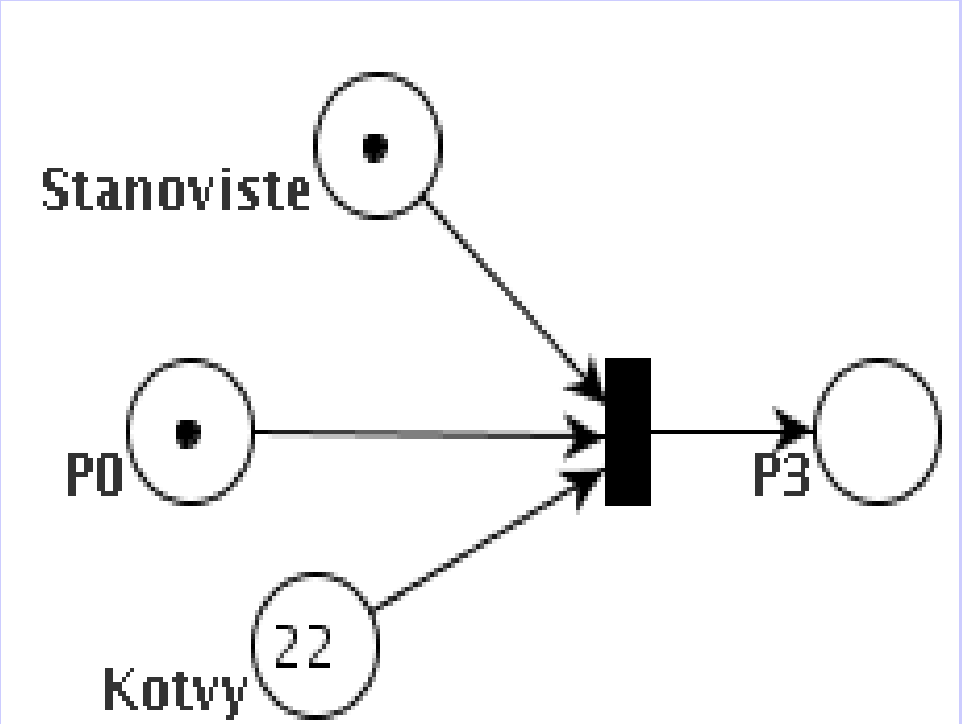
Verifikace modelu

- Kontrola ekvivalence AM-SM
- Verifikace stylu “kouknu a vidim” není správná

Ukázka chybné verifikace (i validace)

- Vlek: zabrání startovního místa, pak kotvy
- realita-AM-SM
- další případy z historie MSI (poruchy stojí ve frontě, hranice, záchody)

```
Seize(stanoviste); // tady
může být přerušen
Enter(kotvy, 1);
....
```



Učebna

```
void akce() {
        Enter(pocitace, 1);
        Wait(Exponential(100));
        Leave(pocitace, 1);
}
void Behavior() {
        opak:
        if (pocitace.Full()) {
            if (Random()<=0.6) akce();
                else {
                    if (Random()<=0.2) {
                        Wait(Uniform(30,60));
                        goto opak;
                    }
                }
        } else
            akce();
    }
```

```
class Gener : public Event {
	void Behavior() {
		(new Student)->Activate();
		Activate(Time+Exponential(10));
	}
};

int main()
{
	Init(0,10000);
	(new Gener)->Activate();
	Run();
	pocitace.Output();
}
```

```
| STORE Stroje                                          |
+-------------------------------------------------------+
|  Capacity = 10  (9 used, 1 free)                      |
|  Time interval = 0 - 10000                            |
|  Number of Enter operations = 863                     |
|  Minimal used capacity = 1                            |
|  Maximal used capacity = 10                           |
|  Average used capacity = 8.42821
  Input queue 'Stroje.Q'
| QUEUE Q                                               |
+-------------------------------------------------------+
|  Time interval = 0 - 10000                            |
|  Incoming  277                                        |
|  Outcoming  277                                       |
|  Maximal length = 9                                   |
|  Average length = 0.667695                            |
|  Minimal time = 0.0142547                             |
|  Maximal time = 110.351                               |
|  Average time = 24.1045                               |
|  Standard deviation = 34.0993                         |
+-------------------------------------------------------+
```

Experimenty

- 50 počítačů
- hledání optima počtu PC
- hledání stavu, kde nikdo nečeká
- statistika – celková doba než se dostane k PC
- pravděpodobnost, že bude plno

```
celkem++;
if (pocitace.Full()) {
      je_plno++;
.....
Print(“Prst: %f”, (float)je_plno/celkem);
```

Vlek

- 2 typy transakcí, stanoviště, kotvy, opakování startu

```
class   Generator : public Event {
public:
        Generator(double interv, int pri) : Event() {
                Interval = interv;
                Pri = pri;
        };
        void    Behavior() {
                (new Lyzar(Pri))->Activate();
                Activate(Time+Exponential(Interval));
        }
        double  Interval;
        int     Pri;
} ;
int     main()
{
        SetOutput("lyzar.dat");
        Init(0, 1000);
        (new Generator(1,0))->Activate();
        (new Generator(10,1))->Activate();
        Run();
```

```
class    Lyzar : public Process {
public: Lyzar(int pri) : Process(pri) {} ;
        void    Behavior() {
                double time = Time;
                int pok=0;
                Seize(Stanoviste);
                opak:
                Enter(Kotvy, 1);
                Wait(Exponential(0.5)); pok++;
                if (Random()<=0.1) {
                        // nezdareny start, kotva jede sama dve cesty (nahoru, dolu)
                        (new KotvaBezi(2))->Activate();
                        goto opak;
                }
                Release(Stanoviste);
                pocetPokusu(pok);
                Wait(jedna_cesta);
                dobaCesty(Time-time);
                (new KotvaBezi(1))->Activate(); // nahore opousti kotvu a ta jede
sama dolu do skladu
        }
};
```

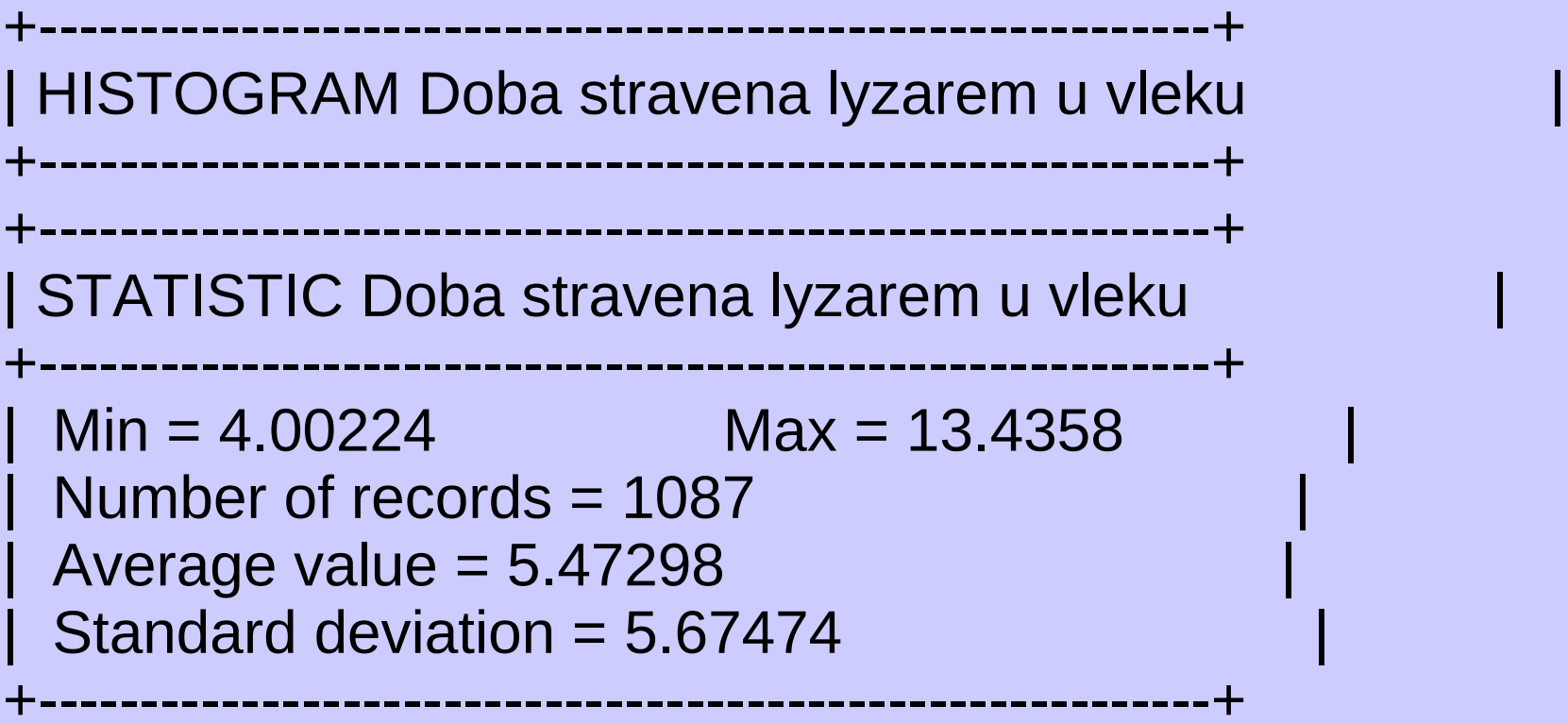
```
// proces volne ujizdejici kotvy
class    KotvaBezi : public Process {
public:
        KotvaBezi(int t) : Process() { T=t; } ;
        // T=1 - jedna cesta, T=2 - cesta tam a zpet

        void    Behavior() {
                Wait(jedna_cesta*T);
                Leave(Kotvy, 1);
                // dojede na seradiste a uvolni kotvu
        }

        int T;
};
```

```
+-----------------------------------------------------+
| STORE Sklad kotev                                   |
+-----------------------------------------------------+
|  Capacity = 40  (16 used, 24 free)                  |
|  Time interval = 0 - 1000                           |
|  Number of Enter operations = 1234                  |
|  Minimal used capacity = 1                          |
|  Maximal used capacity = 24                         |
|  Average used capacity = 10.3968                    |
+-----------------------------------------------------+
```

```
+-----------------------------------------------------+
| FACILITY Stanoviste                                 |
+-----------------------------------------------------+
|  Status = BUSY                                      |
|  Time interval = 0 - 1000                           |
|  Number of requests = 1097                          |
|  Average utilization = 0.617328                     |
+-----------------------------------------------------+
  Input queue 'Stavoviste.Q1'
+-----------------------------------------------------+
| QUEUE Q1                                            |
+-----------------------------------------------------+
|  Time interval = 0 - 1000                           |
|  Incoming  689                                      |
|  Outcoming  682                                     |
|  Maximal length = 12                                |
|  Average length = 1.02976                           |
|  Minimal time = 0.00212281                          |
|  Maximal time = 9.1196                              |
|  Average time = 1.4884                              |
|  Standard deviation = 2.1217                        |
+-----------------------------------------------------+
```


HISTOGRAM Doba stravena lyzarem u vleku
STATISTIC Doba stravena lyzarem u vleku
Min = 4.00224
Max = 13.4358
Number of records = 1087
Average value = 5.47298
Standard deviation = 5.67474

from	to	n	rel	sum
0.000	1.000	0	0.000000	0.000000
1.000	2.000	0	0.000000	0.000000
2.000	3.000	0	0.000000	0.000000
3.000	4.000	0	0.000000	0.000000
4.000	5.000	550	0.505980	0.505980
5.000	6.000	254	0.233671	0.739650
6.000	7.000	128	0.117755	0.857406
7.000	8.000	86	0.079117	0.936523
8.000	9.000	28	0.025759	0.962282
9.000	10.000	23	0.021159	0.983441
10.000	11.000	8	0.007360	0.990800
11.000	12.000	2	0.001840	0.992640
12.000	13.000	5	0.004600	0.997240
13.000	14.000	3	0.002760	1.000000
14.000	15.000	0	0.000000	1.000000

+--+
| HISTOGRAM Pocet pokusu nastoupit |
+--+
+--+
| STATISTIC Pocet pokusu nastoupit |
+--+
| Min = 1   Max = 4 |
| Number of records = 1096 |
| Average value = 1.125 |
| Standard deviation = 1.18118 |
+--+

from	to	n	rel	sum
1.000	2.000	969	0.884124	0.884124
2.000	3.000	118	0.107664	0.991788
3.000	4.000	8	0.007299	0.999088
4.000	5.000	1	0.000912	1.000000
5.000	6.000	0	0.000000	1.000000

Sanitka – příklad s modelem poruchy

- sanitky (2)
- příchody pacientů
- v intervalech Exp(700) dochází k poruše sanitky (každá sanitka má vlastní “časovač”)
- Proces pacienta: přivolá sanitku. Sanitka přijede, naloží, odjede.
- V případě poruchy volá pacient další sanitku

```
void    Behavior() {
            double time = Time;
            int san;
            opak:
            san=-1; prerusen=0;
            for (int a=0; a<Sanitek; a++)
                if (!Sanitka[a].Busy())
                    san=a;
            if (san!=-1)
                Seize(Sanitka[san]); // nasel a bere ji
            else {
                // nenasel, uklada se do fronty
                cekani.Insert(this);
                Passivate();
                goto opak;
            }
            Wait(Uniform(1,5));
            if (prerusen) goto opak; // jestli byl prerusen, opakuje
            Wait(Exponential(1)); // nakladani pacienta
            if (prerusen) goto opak;
```

```
Wait(Uniform(1,5));
 if (prerusen) goto opak;

 Release(Sanitka[san]);

 /*
        Release by mel vytahnout z fronty Q1 proces a ten
   dat do obsluhy. Q1 se ale nepouziva (a proto
        je ve statistikach prazdna).
        Aktivni proces delajici release "rucne" vytahne
           dalsiho z fronty a aktivuje ho.
        Aktivovany se okamzite spusti (naplanuje se na
   cas Time) a pokracuje v miste, kde se pasivoval
 */

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

 hist(Time-time);
```

```
// udalost vyvolani poruchy
class   Porucha : public Event {
public:
        Porucha(int san) : Event() , San(san) {};

        void Behavior() {
                // porucha je tvorena nekolika kroky v case a
musi proto byt implementovana procesem

                (new Oprava(San))->Activate();
                Activate(Time+Exponential(700));
        }

        int San;
} ;
```

```
    Oprava(int san) : Process() , San(san) {};
    void Behavior() {
        Seize(Sanitka[San], 1);

// zabrani sanitky s vyssi prioritou obsluhy (porucha)
// pokud nebyl pred tim nikdo v obsluze, je Q2 prazdna!!, jinak:

        if (Sanitka[San].Q2->Length()>0) {
            // se z ni vyjme process
            Pacient *pac = (Pacient *)Sanitka[San].Q2->GetFirst();

            // nastavi se mu atribut "prerusen"
            pac->prerusen=1;

            // a aktivuje se....
            pac->Activate();
        }

        Wait(Exponential(10)); // probiha oprava
        Release(Sanitka[San]);
    }
```

Opakování experimentů

- Optimalizace
- Zjišťování parametru....

Timeout

- Například ve frontě s netrpělivými požadavky
- Aktivuje se událost, která transakci vyřadí z fronty nebo zruší

```
class Transakce;

class Timeout : public Event {
public:
    Timeout(Transakce *to) : Event() , To(to) {
    };

    void Behavior();

    Transakce *To;
};

void Timeout::Behavior() {
        To->timeout();
        Cancel();
}
```

```
class Transakce : public Process {
public: void Behavior() {
        Timeout *tm = new Timeout(this);
        tm->Activate(Time+15);
        Seize(Linka);
        if (Linka.in == this)
            tm->Cancel();
        else {
            Print("Id %d Timeout.....\n", id);
            return ;
        }
        Wait(Exponential(20));
        Release(Linka);
    }

    void timeout() {
        Out();
        Activate();
    }

};
```

Modely číslicových obvodů

- Bloky
- Zpoždění
- Implementace procesem

```
enum {
    VAL_0,
    VAL_1,
    VAL_X
};

#define DELAY 0.001
```

```
class Nand : public Process {
public:
    Nand() : Process() { OUT=VAL_X; IN[0]=IN[1]=VAL_X; }
    void Behavior() {
        while (1) {
            Passivate();
            if (IN[0]==VAL_X || IN[1]==VAL_X) OUT=VAL_X;
            else OUT = !(IN[0] && IN[1]);
            Print("Time==%f, out=%d\n", Time, OUT);
        }
    }

    void input(int idx, int val) {
        IN[idx]=val;
        Activate(Time+DELAY);
    }

    int OUT;
    int IN[2];
};
```

```
class Value : public Event {
public:
    Value(Nand *n, int idx, int val) : N(n) , Event(), Idx(idx),
Val(val) {}
    void Behavior() {
        N->input(Idx, Val);
        Cancel();
    }

    Nand *N;
    int Idx, Val;
};
```

```
int main()
{
    Init(0, 1);
    Nand *n = new Nand;
    n->Activate();
    (new Value(n, 0, 0))->Activate(0.1);
    (new Value(n, 1, 1))->Activate(0.2);
    (new Value(n, 0, 1))->Activate(0.3);
    Run();
}
```