

## Etapa 3

Program de simulare pentru problema de interferență în condițiile  
în care sistemele sunt prevăzute cu modul de rezervă

Stati Andreea Grupa: 1310A

### 1. Stabilirea modului la care se adaugă rezerva

Scăderea disponibilității este o consecință a suprapunerii efectelor celor două cauze independente de întrerupere accidentală. Pentru a obține o disponibilitate mai ridicată, se adaugă o rezervă identică cu modulul de bază acolo unde întreruperile accidentale afectează într-o măsură mai mare disponibilitatea sistemului. Pentru stabilirea modului la care se adaugă o rezervă trebuie avută în vedere atât frecvența întreruperilor cât și timpul mediu de remediere. Așa cum se cunoaște, pentru modelul primar în care nu intervine fenomenul de interferență, disponibilitatea este dată de relația

$$D = 1/(1 + \lambda A/\mu A + \lambda B/\mu B) \cdot 100 (\%).$$

Relația de calcul evidențiază explicit influența fiecărei cauze de întrerupere accidentală asupra disponibilității sistemului. Pe baza acestei relații se deduce că în ceea ce privește disponibilitatea este mai bine ca **rezerva să se adauge la modulul pentru care raportul  $\lambda/\mu$  este mai mare.**

Cum

$$\lambda A/\mu A = 0.2023/3.6015 = 0.0561$$

si

$$\lambda B/\mu B = 0.1917/2.1835 = 0.0877$$

$$\Rightarrow \lambda A/\mu A < \lambda B/\mu B$$

Deducem ca rezerva se va adăuga la B

## 2. Algoritm de simulare

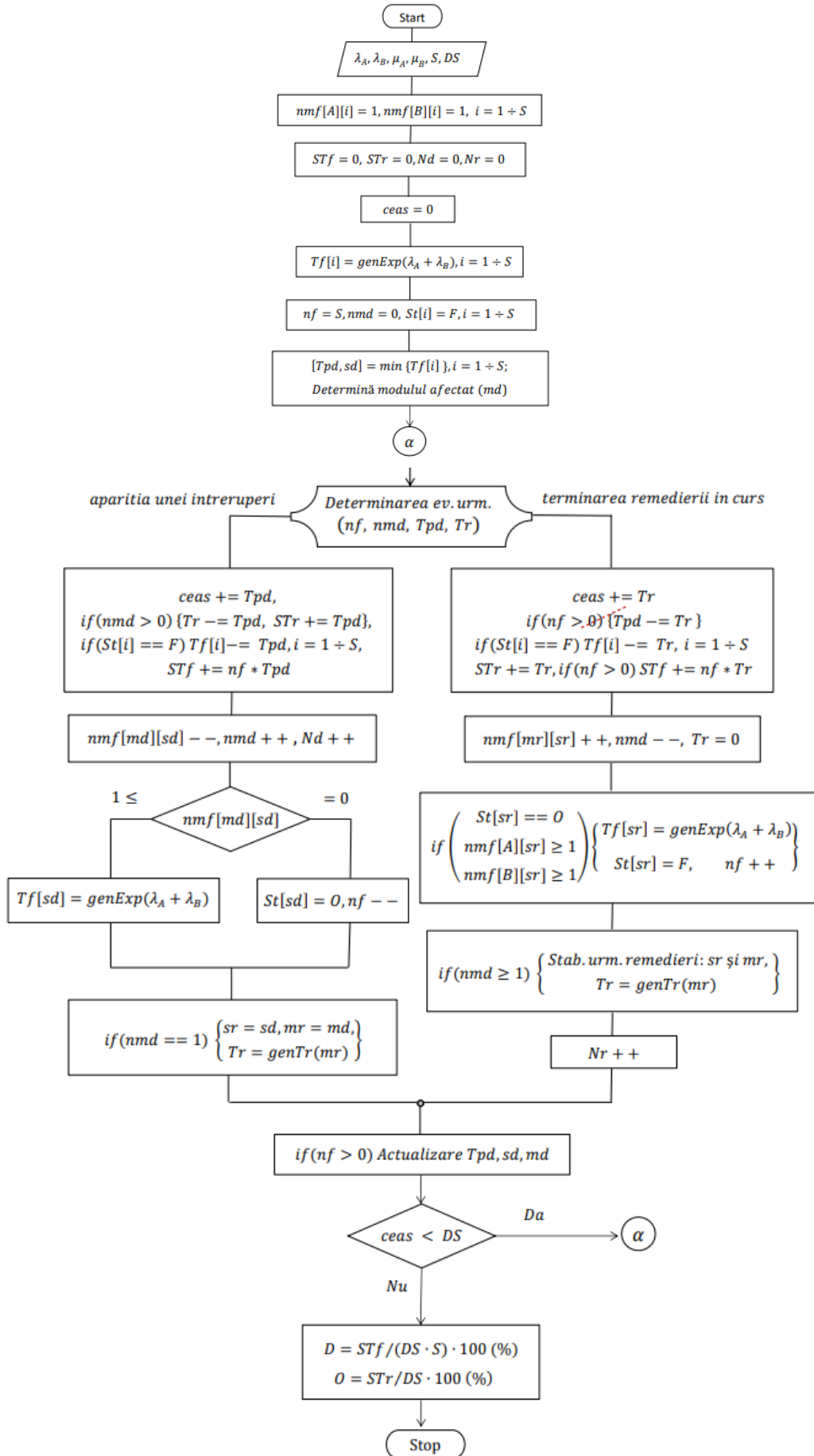
### ➤ O analiză preliminară

La un modul prevăzut cu rezervă, în cazul unei întreruperi accidentale rezerva înlocuiește modulul afectat asigurându-se astfel continuarea funcționării sistemului. Este de dorit ca remedierea modulului defect să se facă înaintea apariției unei noi întreruperi pentru a readuce sistemul la starea de toleranță de la început. Ca urmare, la această problemă de interferență nu orice întrerupere duce și la oprirea sistemului. De asemenea, nu orice remediere permite repornirea sistemului. În aceste condiții, algoritmul de simulare trebuie să urmărească explicit schimbările de stare din sistem, pentru ca la apariția unei întreruperi să se poată stabili dacă sistemul se oprește sau își continuă funcționarea cu modulul de rezervă, și dacă după remedierea unui modul afectat de întrerupere sistemul poate fi repus în funcțiune sau nu.

### ➤ Semnificația mărimilor folosite în algoritmul de simulare:

- $St[i]$ ,  $i = 1 \div S$  – starea sistemului  $i$ : în funcțiune ( $F$ ) sau oprit ( $O$ );
- $nmf[m][i]$ ,  $m = A$  sau  $B$ ,  $i = 1 \div S$  – numărul modulelor funcționale de tip  $A$  sau  $B$  la sistemul  $i$  (la inițializare se specifică astfel modulul care este prevăzut cu rezervă);
- $nf$  – numărul de sisteme în funcțiune la un moment dat;
- $nmd$  – numărul de module afectate de întrerupere care necesită remediere, indiferent de tipul lor sau de sistemele de care aparțin.
- $Tf[i]$  – timpul de funcționare până la prima întrerupere accidentală la sistemul  $i$ ,  $i = 1 \div S$ ; locația  $Tf[i]$  nu are semnificație când sistemul  $i$  este oprit ( $St[i] = O$ ).
- $Tpd$  – timpul până la prima întrerupere accidentală (defectare) care se va produce, indiferent de tipul modulului afectat sau de sistemul de care aparține; variabila nu are semnificație când  $nf = 0$ .
- $Tr$  – timpul până la terminarea remedierii în curs; variabila nu are semnificație când  $nmd = 0$ ;
- $sd$ ,  $md$  – sistemul la care va apărea prima defectare și tipul modulului afectat;
- $sr$ ,  $mr$  – sistemul la care se efectuează o operație de remediere și tipul modulului în curs de remediere. Statistici:
- $STf$  – Suma timpilor de funcționare pentru cele  $S$  sisteme în perioada simulată;
- $STr$  – Suma timpilor de lucru pentru operațiile de remediere efectuate de muncitor în perioada simulată;
- $Nd$  – Numărul de întreruperi accidentale (defectări) produse în perioada de monitorizare;
- $Nr$  – Numărul de remedieri efectuate de muncitor în perioada de simulare.

### ➤ Precizare: Pentru început vom considera că modulul de rezervă se menține în stare pasivă, nefiind așadar solicitat cât timp nu este folosit. Algoritmul de simulare este prezentat sub formă de schemă logică în figura următoare:



### 3. Verificarea programului de simulare

- Cu inițializarea  $nmf[A][i] = 1$ ,  $nmf[B][i] = 1$ ,  $i = 1 \div S$ , se obține o altă implementare pentru problema de interferență a sistemelor fără modul de rezervă, studiată la Etapa 2. Trebuie să se verifice că prin cele două programe de simulare se obțin aceleași rezultate.
- Verificările făcute la etapa anterioară și aspectele legate de precizia estimării sunt valabile și în acest caz. Observație: Modulele de rezervă sunt menținute în stare pasivă și nu sunt afectate de întreruperi cât timp nu sunt folosite. Prin urmare, rata medie de întrerupere accidentală pentru un sistem în funcțiune este tot  $\lambda A + \lambda B$  cât era și la etapa anterioară.

### 4. Completarea programului de simulare pentru a acoperi și alte aspecte

#### Codul sursă

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
using namespace std;

enum Modul { A, B };
enum Stare { O, F };

double genExp(double lambda)
{
    double u, x;
    u = (double)rand() / (RAND_MAX + 1);
    x = -1 / lambda * log(1 - u);
    return x;
}

double genTr(Modul m, double miuA, double miuB)
{
    double Tr;
    if (m == B)
    {
        Tr = genExp(miuB);
    }
    else
    {
        Tr = genExp(miuA);
    }
    return Tr;
}

int main(void)
{
    double lambdaA = 0.2023;
    double lambdaB = 0.1917;
```

```

cout << "lambdaA = " << lambdaA << endl;
cout << "lambdaB = " << lambdaB << endl;
cout << endl;

double miuA = 3.6015;
double miuB = 2.1835;
cout << "miuA = " << miuA << endl;
cout << "miuB = " << miuB << endl;
cout << endl;

/* probabilitatea ca la un sistem oprit modulul care
   necesita remediere sa fie de tip A si respectiv, de tip B. */
double PA = lambdaA / (lambdaA + lambdaB);
double PB = lambdaB / (lambdaA + lambdaB);
cout << "PA = " << PA << endl;
cout << "PB = " << PB << endl;
cout << "PA + PB = " << PA + PB << endl;
cout << endl;

double Oc = 0, D;

int S;
for (S = 1; S <= 8; S++)
{
    cout << "\nNr. sisteme=" << S << endl;
    double NS = 10000000;
    double DS;

    double STr = 0;
    double STf = 0;
    int Nd = 0;
    int Nr = 0;

    double ceas = 0;

    int nmf[2][100];
    for (int i = 1; i <= S; i++)
    {
        nmf[A][i] = 1;
        nmf[B][i] = 2;
    }

    int nf = S;
    int nmd = 0;

    double Tf[100];
    for (int i = 1; i <= S; i++)
    {
        Tf[i] = genExp(lambdaA + lambdaB);
    }

    Stare St[100];
    for (int i = 1; i <= S; i++)
    {
        St[i] = F;
    }

    double Tpd;
    int sd;

```

```

Modul md;

int sr;
Modul mr;
double Tr = 0;

/* [Tpd, sd] = min{Tf[i]}; i = 1 - S
   determina modulul afectat md
   Tpd = timpul pana la prima intrerupere accidentala (defectare)
care se va produce, indiferent de
       tipul modulului afectat sau de sistemul de care
apartine; variabila nu are semnificatie cand nf = 0.
   sd, md = sistemul la care va aparea prima defectare si tipul
modulului afectat;
   nf = nr de sisteme in functiune la un moment dat
*/
double min = 10000000;
int ind;
for (int i = 1; i <= S; i++)
{
    if (St[i] == F && Tf[i] < min)
    {
        min = Tf[i];
        ind = i;
    }
}
Tpd = min;
sd = ind;
double u = (double)rand() / RAND_MAX;
if (u < PA)
{
    md = A;
}
else
{
    if (u < (PA + PB))
        md = B;
}
do {
    if (nmd == 0 || ((nf > 0) && Tpd < Tr)){ // aparitia unei
intreruperi

        ceas += Tpd;
        if (nmd > 0)
            Tr -= Tpd;

        for (int i = 1; i <= S; i++)
            if (St[i] == F)
                Tf[i] -= Tpd;

        STf += nf * Tpd;
        nmf[md][sd]--;
        nmd++;
        Nd++;

        if (nmf[md][sd] == 0)
        {
            St[sd] = 0;
            nf--;
        }
    }
}

```

```

else
    if (nmf[md][sd] > 0)
        Tf[sd] = genExp(lambdaA + lambdaB);
if (nmd == 1)
{
    sr = sd;
    mr = md;
    Tr = genTr(mr, miuA, miuB);
    STr += Tr;
}
}
// terminarea remedierii in curs
else
{
    Nr++;
    ceas += Tr;
    for (int i = 1; i <= S; i++)
        if (St[i] == F)

            Tf[i] -= Tr;
    STf += nf * Tr;
    nmf[mr][sr]++;
    nmd--;

    if (St[sr] == 0 && nmf[A][sr] >= 1 && nmf[B][sr] >=
1)
    {
        St[sr] = F;
        nf++;
        Tf[sr] = genExp(lambdaA + lambdaB);
    }

    if (nmd > 0)
    {
        for (int i = 1; i <= S; i++)
        {
            if (St[i] == 0)
            {
                if (nmf[A][i] < 1)
                {
                    mr = A;
                    sr = i;
                    break;
                }
                else
                {
                    if (nmf[B][i] < 2)//if
(nmf[B][i] < 1)
                    {
                        mr = B;
                        sr = i;
                        break;
                    }
                }
            }
        }
        Tr = genTr(mr, miuA, miuB);
        STr += Tr;
    }
}
}

```

```

        if (nf > 0)
        {
            min = 100000000;
            for (int i = 1; i <= S; i++)
            {
                if (Tf[i] < min && St[i] == F)
                {
                    min = Tf[i];
                    ind = i;
                }
            }
            Tpd = min;
            sd = ind;
            double u = (double)rand() / RAND_MAX;
            if (u < PA)
            {
                md = A;
            }
            else
            {
                if (u < (PA + PB))
                    md = B;
            }
        }
    } while (Nd < NS);
    DS = ceas;
    D = (STf / (DS * S)) * 100;
    Oc = (STr / DS) * 100;
    cout << "Disponibilitate=" << D << endl;
    cout << "Grad ocupare=" << Oc << endl;
}
return 0;
}

```

### **Output:**

lambdaA = 0.2023  
lambdaB = 0.1917

miuA = 3.6015  
miuB = 2.1835

PA = 0.513452  
PB = 0.486548  
PA + PB = 1

Nr. sisteme=1  
Disponibilitate=93.4936  
Grad ocupare=13.4614

Nr. sisteme=2  
Disponibilitate=90.511  
Grad ocupare=26.055



Nr. sisteme=3  
Disponibilitate=89.0987  
Grad ocupare=38.463

Nr. sisteme=4  
Disponibilitate=87.4954  
Grad ocupare=50.3744

Nr. sisteme=5  
Disponibilitate=85.6244  
Grad ocupare=61.6322

Nr. sisteme=6  
Disponibilitate=83.3401  
Grad ocupare=71.9435

Nr. sisteme=7  
Disponibilitate=80.5695  
Grad ocupare=81.1724

Nr. sisteme=8  
Disponibilitate=77.155  
Grad ocupare=88.8323

#### 4.1 Generalizare privind regimul de lucru pentru modulul de rezervă

În funcție de gradul de solicitare a rezervei (în sensul de modul cu regim de rezervă), aceasta poate fi pasivă, activă sau parțial activă. Pentru o rezervă identică cu modulul de bază, rata întreruperilor accidentale (rata de defectare) se exprimă cu relația:

$\lambda R = \alpha \lambda M$ , în care  $M = A$  sau  $B$ , iar  $\alpha \in [0, 1]$ . (1)

- $\alpha = 0 \rightarrow$  rezervă pasivă (nesolicitată cât timp nu este în funcțiune);
- $\alpha = 1 \rightarrow$  rezervă activă (solicitată în aceeași măsură cu modulul de bază);
- $\alpha \in (0, 1) \rightarrow$  rezervă parțial activă (solicitată într-o măsură mai mică decât modulul de bază);

În studiul nostru vom lucra cu  $\alpha = 0.5$  și vom numi rezerva ca fiind semi-activă.

➤ Rezervă la modulul  $B$

$$\lambda = \lambda A + \lambda B(1 + \alpha(nmf[B][i] - 1))$$

Probabilitatea ca la sistemul  $i$  modulul care se va defecta mai întâi să fie de tip  $A$  este:

$$pA = \lambda A / (\lambda A + \lambda B(1 + \alpha(nmf[B][i] - 1)))$$

## 4.2 Alegerea următorului modul pentru remediere


La terminarea remedierii în curs, dacă mai sunt module defecte, muncitorul trebuie să înceapă imediat o nouă remediere. La alegerea următorului modul care să fie remediat, muncitorul trebuie să urmărească repunerea cât mai rapidă în funcțiune a unui sistem oprit. Modulele de rezervă defecte de la sistemele în funcțiune trebuie remediate ulterior. Prin urmare, la alegerea următorului modul pentru remediere trebuie să se țină cont de starea sistemelor și de intensitățile medii de remediere pentru cele două tipuri de module,  $A$  și  $B$ .

➤ Pentru un sistem cu rezervă la modulul  $B$

- a) Dacă  $\mu_A > \mu_B$  ordinea de prioritate la remediere este:  $S2$  sau  $S3$  (se repară  $A$ ),  $S4$ ,  $S1$ ;  
b) Dacă  $\mu_B > \mu_A$  ordinea de prioritate la remediere este:  $S4$ ,  $S2$  sau  $S3$  (se repară  $A$ ),  $S1$ .

Exemplificare pentru cazul cu rezervă la modulul  $B$  și  $\mu_A > \mu_B$ .

sisteme	1	2	3	4	5	6	7	8	9	10
$nmf:$										
$A$	1	1	1	1	1	0	1	0	1	1
$B$	2	2	1	0	2	2	1	1	2	0
prioritate	-	-	3	2	-	1	3	1	-	2



$sr = 6, mr = A$

## 4.3 Întreruperea remedierii în curs

Dacă în timp ce muncitorul remediază o rezervă de la un sistem în funcțiune se produce o întrerupere accidentală care conduce la oprirea sistemului, se poate pune problema întreruperii remedierii în curs pentru a interveni cu prioritate asupra modulului afectat, astfel încât sistemul oprit să fie repus în funcțiune cât mai repede.

### Codul sursa

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
```

```

#define alfa 1

using namespace std;

enum Module { A, B };
enum Stare { 0, F };

double genExp(double lambda)
{
    double u, x;
    u = (double)rand() / (RAND_MAX + 1);
    x = -1 / lambda * log(1 - u);
    return x;
}

double genTr(Module m, double miuA, double miuB)
{
    double Tr;
    if (m == B)
    {
        Tr = genExp(miuB);
    }
    else
    {
        Tr = genExp(miuA);
    }
    return Tr;
}

int main(void)
{
    double lambdaA = 0.2023;
    double lambdaB = 0.1917;

    double miuA = 3.6015;
    double miuB = 2.1835;

    double Oc = 0, D;

    int S;
    for (S = 1; S <= 8; S++)
    {
        cout << "\nNr. sisteme=" << S << endl;
        double NS = 100000000;
        double DS;

        double STr = 0;
        double STf = 0;
        int Nd = 0;
        int Nr = 0;

        double ceas = 0;
    }
}

```

```

int nmf[2][100];

int nf = S;
int nmd = 0;

double Tf[100];

Stare St[100];

double Tpd;
int sd;
Module md;

double Tr = 0;
int sr;
Module mr;
for (int i = 1; i <= S; i++)
{
    nmf[B][i] = 2;//nmf[B][i] = 1;
    nmf[A][i] = 1;
}
for (int i = 1; i <= S; i++)
{
    St[i] = F;
    Tf[i] = genExp(lambdaA + lambdaB * (1 + alfa * (nmf[B][i]
- 1)));
}
double min = 100000000;
int ind;
for (int i = 1; i <= S; i++)
{
    if (St[i] == F && Tf[i] < min)
    {
        min = Tf[i];
        ind = i;
    }
}
Tpd = min;
sd = ind;
double u = (double)rand() / RAND_MAX;
double pA = lambdaA / (lambdaA + lambdaB * (1 + alfa *
(nmf[B][sd] - 1)));
double pB = 1 - pA;
if (u < pA)
{
    md = A;
}
else
{
    if (u < (pA + pB))
        md = B;
}

```

```

    }
    do {
        if (nmd == 0 || ((nf > 0) && Tpd < Tr))
        {
            ceas += Tpd;
            if (nmd > 0)
            {
                Tr -= Tpd;
            }
            for (int i = 1; i <= S; i++)
            {
                if (St[i] == F)
                {
                    Tf[i] -= Tpd;
                }
            }
            STf += nf * Tpd;
            nmf[md][sd]--;
            nmd++;
            Nd++;
            if (nmf[md][sd] == 0)
            {
                St[sd] = 0;
                nf--;
            }
            else
            {
                if (nmf[md][sd] >= 1)
                    Tf[sd] = genExp(lambdaA + lambdaB * (1 +
alfa * (nmf[B][sd] - 1)));
                else
                {
                    St[sd] = 0;
                    --nf;
                }
            }
        }
        if ((nmd == 1) || (St[sr] == F && St[sd] == 0) ||
(sd == sr && md == B))
        {
            sr = sd;
            mr = md;
            if (nmd > 1)
            {
                STr -= Tr;
            }
            Tr = genTr(mr, miuA, miuB);
            STr += Tr;
        }
    }
    else
    {
        Nr++;
    }
}

```

```

ceas += Tr;
for (int i = 1; i <= S; i++)
{
    if (St[i] == F)
    {
        Tf[i] -= Tr;
    }
}
STf += nf * Tr;
nmf[mr][sr]++;
nmd--;
if (nmf[A][sr] >= 1 && nmf[B][sr] >= 1)
{
    if (St[sr] == 0)
    {
        St[sr] = F;
        nf++;
    }
    Tf[sr] = genExp(lambdaA + lambdaB * (1 + alfa
* (nmf[B][sd] - 1)));
}
if (nmd > 0)
{
    bool gasit = false;
    for (int i = 1; i <= S; i++)
    {
        if (nmf[A][i] == 0)
        {
            mr = A;
            sr = i;
            gasit = true;
            break;
        }
    }
    if (gasit == false)
    {
        for (int i = 1; i <= S; i++)
        {
            if (nmf[B][i] == 0)
            {
                mr = B;
                sr = i;
                gasit = true;
                break;
            }
        }
    }
    if (gasit == false)
    {
        for (int i = 1; i <= S; i++)
        {
            if (nmf[A][i] == 1 && nmf[B][i] ==

```

1)

```

        {
            mr = B;
            sr = i;
            gasit = true;
        }
    }
    Tr = genTr(mr, miuA, miuB);
    STr += Tr;
}
}
if (nf > 0)
{
    min = 100000000;
    for (int i = 1; i <= S; i++)
    {
        if (Tf[i] < min && St[i] == F)
        {
            min = Tf[i];
            ind = i;
        }
    }
    Tpd = min;
    sd = ind;
    double u = (double)rand() / RAND_MAX;
    double pA = lambdaA / (lambdaA + lambdaB * (1 + alfa
* (nmf[B][sd] - 1)));
    double pB = 1 - pA;
    if (u < pA)
    {
        md = A;
    }
    else
    {
        if (u < (pA + pB))
            md = B;
        }
    }
} while (Nd < NS);
DS = ceas;
D = (STf / (DS * S)) * 100;
Oc = (STr / DS) * 100;
cout << "Disponibilitate=" << D << endl;
cout << "Grad ocupare=" << Oc << endl;
}
return 0;
}
```

## Rezultate:

Stări	Fără rezerva		Cu rezerva activa $\alpha=1$		Cu rezerva semi activă $\alpha=0.5$		Cu rezerva pasiva $\alpha=0$	
	D(%)	O(%)	D(%)	O(%)	D(%)	O(%)	D(%)	O(%)
S1	93.4936	13.4614	92.4927	20.312	92.9769	17.021	93.4927	13.4604
S2	90.511	26.055	92.0721	37.8278	92.4536	32.8779	92.95	26.7624
S3	89.0987	38.463	91.0352	55.0587	91.6093	48.4298	92.2939	39.8584
S4	87.4954	50.3744	89.6508	70.2334	90.4713	62.9663	91.4472	52.6558
S5	85.6244	61.6322	87.6985	82.5929	88.8594	75.8801	90.2972	64.9991
S6	83.3401	71.9435	84.9088	91.4836	86.4515	86.5545	88.6264	76.5456
S7	80.5695	81.1724	81.1117	96.7427	82.9741	93.9676	85.9677	86.5965
S8	77.155	88.8323	76.5214	99.0753	78.1889	98.0287	81.73	94.1506

## Concluzii:

Se observa cum prin adăugarea unei rezerve, atât disponibilitatea ,cat si gradul de ocupare, cresc semnificativ, mai ales in cazul rezervei pasive.