Sisteme de Operare

- Paradigme ale programării concurente
- Elemente de blocaj:
  - Alocarea resurselor
  - Tratarea blocajelor

Problema cititori/scriitori
soluția cu prioritatea cititorilor asupra scriitorilor

```
nt nrcit =0;
semafor s, scriere;
s =scriere =1 ;
cititor()
{
.....
   P(s);
   nrcit =nrcit +1;
   if(nrcit = =1)  P(scriere);
   V(s);
.....//lucrul cu resurse
   P(s);
   nrcit =nrcit – 1;
   if(nrcit = =0) V(scriere);
   V(s);
.....

   }
```

p. intrare

p. iesire

SC

```
scriitor()
{
.....
   P(scriere);
.....//lucru cu resurse
   V(scriere);
.....

   }
```

Problema cititori/scriitori
soluția cu prioritatea scriitorilor asupra cititorilor

```
int nrcit =nrscr =0;
semafor s, s1, s2, citire, scriere;
s =s1 =s2 =citire =scriere =1 ;
```

```
cititor_i()
{
.....
  P(s2);
  P(citire);
  P(s);
  nrcit =nrcit +1;
  if(nrcit = =1) P(scriere);
  V(s);
  V(citire);
  V(s2);
.....          //lucrul cu resursa
  P(s);
  nrcit =nrcit –1;
  if(nrcit==0) V(scriere);
  V(s);
}
```

```
scriitor_j()
{
.....
  P(s1);
  nrscr =nrscr +1;
  if(nrscr = =1) P(citire);
  V(s1);
  P(scriere);
.....          //lucrul cu resursa
  V(scriere);
  P(s1);
  nrscr =nrscr – 1;
  if(nrscr = =0)  V(citire);
  V(s1);
.....

}
```

Problema cititori/scriitori
soluția cu prioritatea scriitorilor asupra cititorilor

- Este garantat faptul că în momentul execuţiei de către primul cititor a unui **V(citire)**, în şirul de aşteptare a variabilei **citire** se află cel mult procese cititori (dintre care primul va fi blocat).
- Accesul la semaforul **citire** trebuie făcut într-o secţiune critică între **P(s2)** şi **V(s2)**. Astfel următorii cititori se vor bloca pe **s2** şi nu pe **citire**.
- Semaforul **s2** asigură execuţia unui singur cititor în secţiunea critică dată de semaforul **citire**.
- Citire asigură blocarea cititorului dacă există un scriitor blocat pe **P(citire)**
- semaforul **s**, asigură excluderea la prelucrarea variabilei comune **nrcit** şi este folosit numai când sunt procese care citesc.

Problema cititori/scriitori
soluția cu prioritatea scriitorilor asupra cititorilor

- implementare prin transmitere de mesaje
  - Dacă Count > 0 nici un scriitor nu aşteaptă; putem avea mai mulţi cititori;
  - Dacă Count = 0 avem un scriitor care aşteaptă;
  - Dacă Count < 0 avem cereri de scriere care aşteaptă să se termine citirile.

Problema cititori/scriitori
soluția cu prioritatea scriitorilor asupra cititorilor

```
void reader(int i)
{
   message rmsg;
      while (true) {
         rmsg = i;
         send (readrequest, rmsg);
         receive (mbox[i], rmsg);
         READUNIT ();
         rmsg = i;
         send (finished, rmsg);
      }
}
void writer(int j)
{
   message rmsg;
   while(true) {
      rmsg = j;
      send (writerequest, rmsg);
      receive (mbox[j], rmsg);
      WRITEUNIT ();
      rmsg = j;
      send (finished, rmsg);
   }
}
```

```
void controller()
{
   while (true)
   {
      if (count > 0) {
         if (!empty (finished)) {
            receive (finished, msg);
            count++;
         }
         else if (!empty (writerequest)) {
            receive (writerequest, msg);
            writer_id = msg.id;
            count = count - 100;
         }
         else if (!empty (readrequest)) {
            receive (readrequest, msg);
            count--;
            send (msg.id, "OK");
         }
      }
      if (count == 0) {
         send (writer_id, "OK");
         receive (finished, msg);
         count = 100;
      }
      while (count < 0) {
         receive (finished, msg);
         count++;
      }
   }
}
```

- Sursa: William Stallings - Operating Systems: Internals and Design Principles, 6th Edition, Prentice Hall, 2008

Problema bărbierului

- Un bărbier are o sală de aşteptare cu **n** scaune. Dacă nu are clienţi bărbierul doarme. Dacă un client care intră nu găseşte loc să se aşeze pleacă, iar dacă este liber un scaun se aşează pe el. Atât timp cât există cel puţin un client aşezat pe scaunul bărbierului, bărbierul lucrează.

Problema bărbierului

```
#define CHAIR 5     //scaune pentru clienţi
semaphore client=0;
// numărul de clienţi care aşteaptă să fie serviţi
semaphore bărbier=1;
//numărul de bărbieri care aşteaptă clienţi
semaphore mutex=1;
// semafor pt excluderea mutuală
int waiting=0;   // clienţii aşteaptă să fie serviţi
void barber( )
{
  while(true){
    P(client); // daca numărul de clienţi este 0 bărbierul doarme
    P(mutex);  // acces la variabila waiting
    waiting --;   //scade numărul de clienţi care aşteaptă
    V(mutex);  //eliberare scaun = iau clientul
    Bărbiereşte( );
    V(bărbier);   //bărbier gata de lucru
  }
}
```

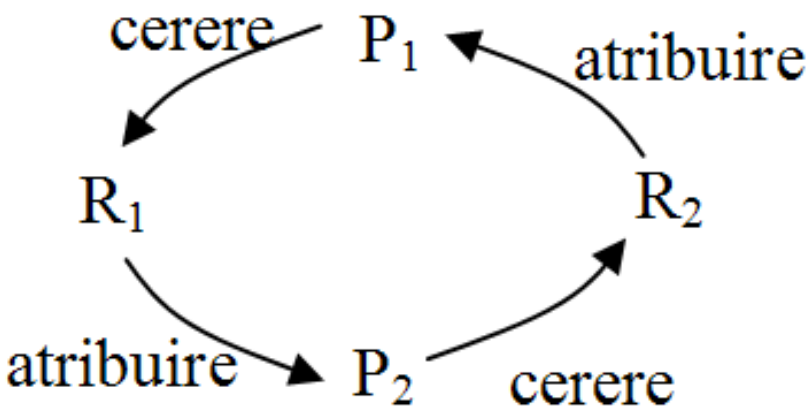
```
void client( )
{
  P(mutex);  //intrare în regiunea critică
  if (waiting< CHAIR){
        // dacă nu sunt scaune libere pleacă
   waiting++;      // creşte numărul clienţilor
   V(client);
        // dacă este cazul trezeşte bărbierul
   V(mutex);
      //eliberează accesul  pentru un nou client ce poate intra
    P(bărbier);
      //aşteaptă să se elibereze un bărbier
    este_bărbierit( );  //este servit
  }else{
   V(mutex);       //nu sunt scaune libere
  }
}
```

Elemente de blocaj

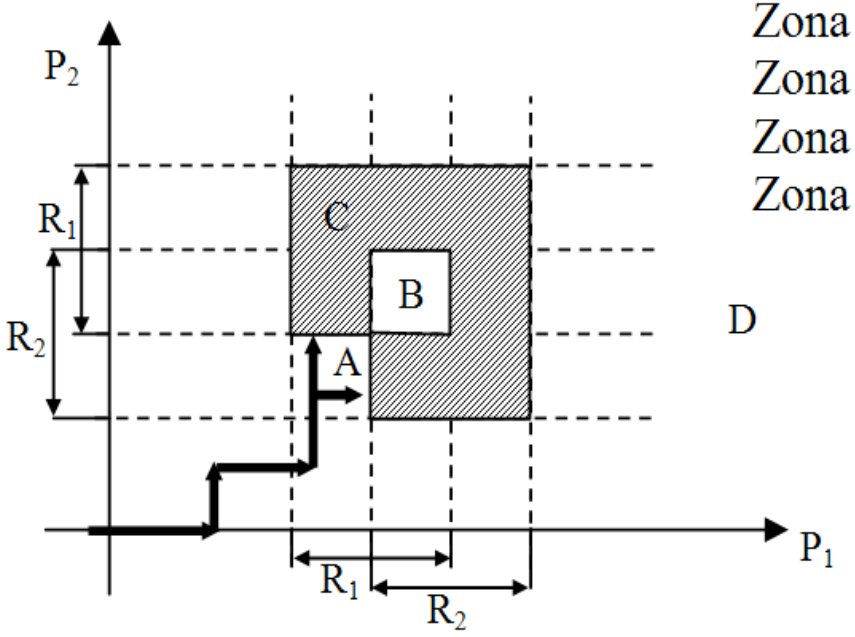
- Se numeşte **blocaj** situaţia în care o resursă cerută de un proces este menţinută în starea ocupat de către alt proces aflat la rândul lui în aşteptarea eliberării unei resurse. Etapele parcurse de un proces pentru obţinerea unei resurse sunt:
  - **cerere de acces** – dacă cererea nu este satisfăcută imediat, procesul este nevoit să aştepte;
  - **utilizare** – procesul poate folosi resursa;
  - **eliberare** – procesul eliberează resursa.
- Resursele pot fi:
  - **reutilizabile** (utilizate de un proces şi apoi eliberate pentru a putea fi utilizate de alte procese – timp CPU, canale I/O, memoria principală şi virtuală , fişiere, baze de date, semafoare)
  - **consumabile** (întreruperi, semnale, mesaje, informaţii din buffer-ele de I/O).

Elemente de blocaj

- Se spune că un set de procese se află în starea de **interblocare** atunci când oricare proces din set se află în aşteptarea unui eveniment ce poate fi produs numai de către un alt proces din setul respectiv.



Elemente de blocaj

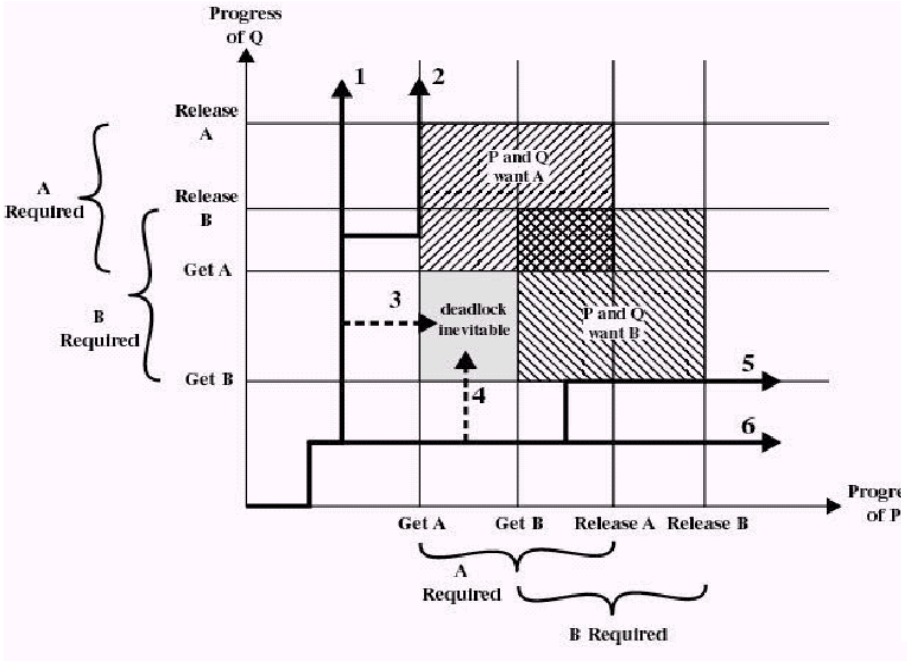


Zona A este zonă de *potențial blocaj* (*zonă nesigură*) ;
Zona B este o zonă ce nu poate fi atinsă ;
Zona C este *zonă de blocaj* dacă se vine din zona A ;
Zona D este *zonă liberă* de blocaj.

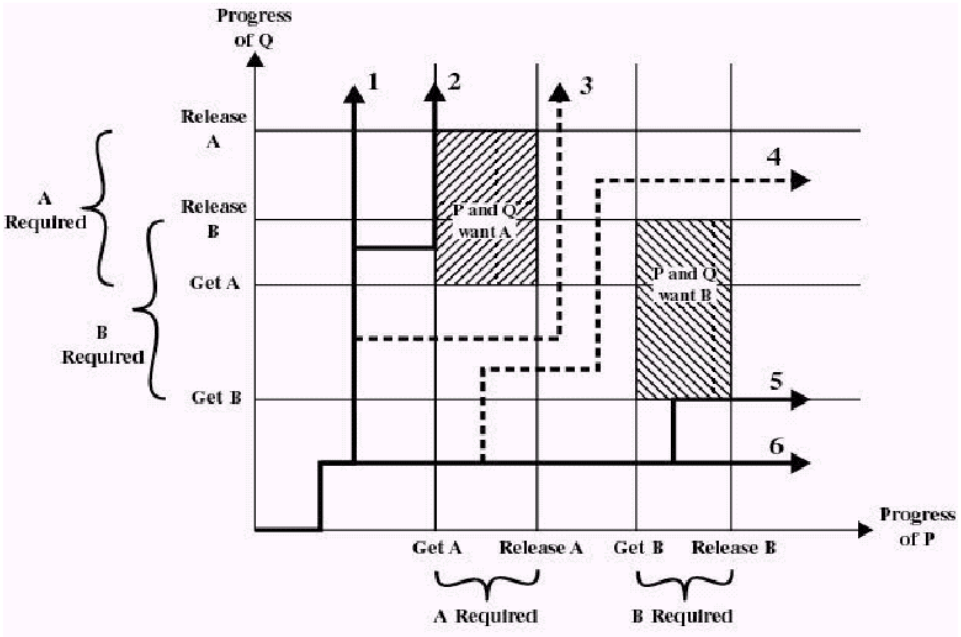
Elemente de blocaj

- Interblocarea apare în sistem dacă şi numai dacă sunt îndeplinite simultan următoarele condiţii:
  - **excluderea mutuală**: există cel puţin o resursă ocupată în mod exclusiv de către un proces ;
  - **ocupare şi aşteptare**: există cel puţin un proces care menţine ocupată cel puţin o resursă critică şi aşteaptă să obţină resurse suplimentare ocupate în acel moment de către alte procese ;
  - **imposibilitatea achiziţionării forţate**: resursele nu pot fi achiziţionate forţat de către un proces de la procesul care le ocupă în momentul respectiv şi sunt eliberate numai de către procesele care le ocupă după terminarea sarcinilor ;
  - **aşteptare circulară**.

Elemente de blocaj



Rularea proceselor P şi Q cu blocaj



Rularea proceselor P şi Q fără blocaj

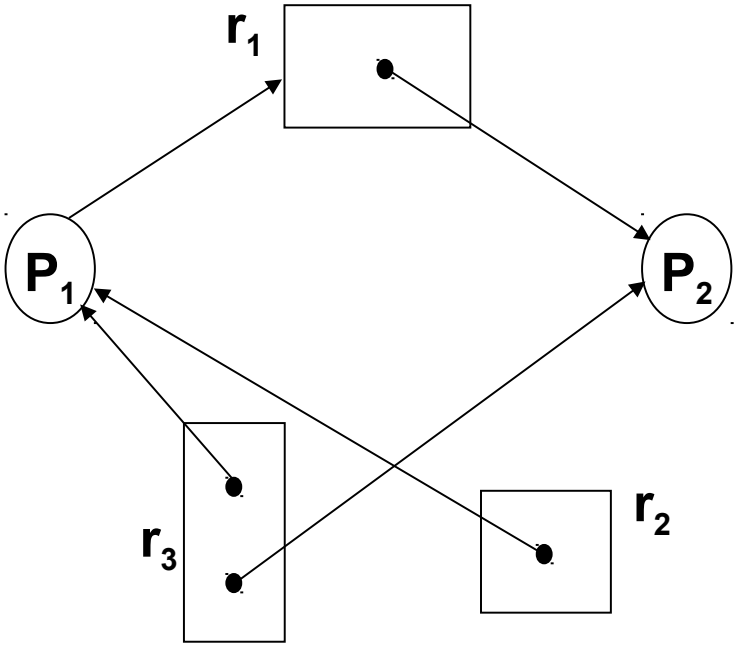
Graful de alocare a resurselor

- **Definiţie**: Starea de interblocare poate fi descrisă prin folosirea unui graf orientat numit graf de alocare a resurselor sistemului. Acesta este format dintr-o pereche **G=(N, A)** unde **N** reprezintă un set de noduri şi **A** un set de arce.
- Setul de noduri conţine două mulţimi:
  - **P = (p1, p2, … pn)** – setul care conţine toate procesele din sistem şi
  - **R = (r1, r2, … rn)** – setul care conţine toate tipurile de resurse din sistem.

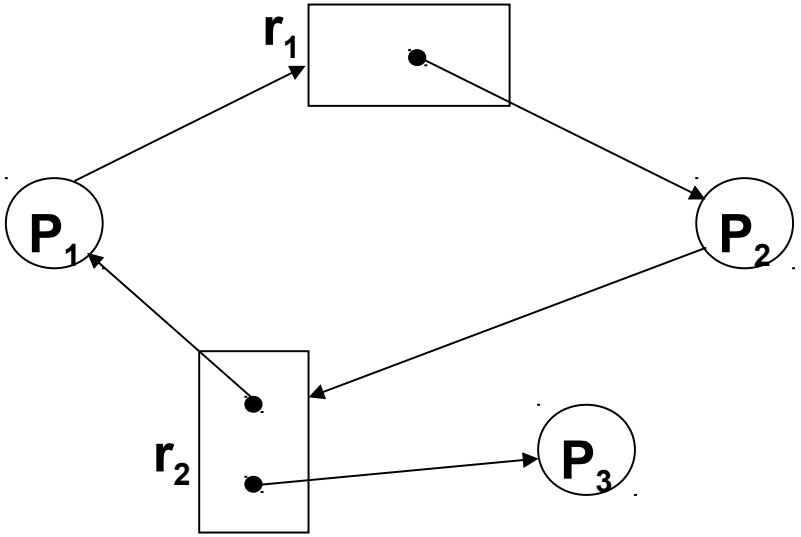
Graful de alocare a resurselor

- Fiecare element din setul **A** de arce reprezintă o pereche ordonată **(p_i, r_j)** sau **(r_j, p_i)** în care **p_i** este un proces din **P** iar **r_j** este o resursă din **R**.
- Dacă **(p_i, r_j)** $\in$ **A** atunci există un arc orientat de la procesul **p_i** la tipul de resursă **r_j** , ceea ce înseamnă că procesul **p_i** a formulat o cerere pentru un element al tipului de resursă **r_j** şi aşteaptă obţinerea ei.
- Dacă **(r_j, p_i)** $\in$ **A** atunci există un arc orientat de la tipul de resursă **r_j** la procesul **p_i**, ceea ce înseamnă că procesului **p_i** i-a fost alocat un element al tipului de resursă **r_j** .
- Un arc de tipul **(p_i, r_j)** se numeşte arc cerere şi **(r_j, p_i)** se numeşte arc de alocare.

Graful de alocare a resurselor



Graf de alocare a resurselor fără interblocare



Graf de alocare a resurselor cu interblocare

Graful de alocare a resurselor

- Conform definiţiei anterioare, **dacă graful nu conţine bucle atunci în sistem nu există interblocare**. Dacă apare o singură buclă, interblocarea poate să apară.
- **Dacă fiecare tip de resursă este format dintr-un singur element atunci existenţa unei bucle în cadrul grafului arată că în sistem a apărut o interblocare**, fiecare proces implicat fiind în această stare.
- **Dacă fiecare tip de resursă conţine mai multe elemente atunci existenţa unei bucle în cadrul grafului nu implică în mod necesar apariţia interblocării** (este o condiţie necesară nu şi suficientă).

Graful de alocare a resurselor

- O stare se numeşte **sigură** dacă resursele pot fi alocate proceselor (fiecăruia în parte) până la nivelul cererii într-o ordine oarecare şi cu evitarea blocării.
- O mulţime de procese este într-o stare sigură dacă există o **secvenţă sigură**.
- O secvenţă de procese **< p_1, p_2, ... p_n >** se numeşte **sigură** pentru starea de alocare dacă pentru fiecare **p_i** pot fi atribuite resursele cerute din cele disponibile plus cele disponibilizate de **p_j** cu **j < i**.
- Dacă resursele de care are nevoie **p_i** nu sunt imediat disponibile toate, atunci **p_i** aşteaptă până se termină **p_j** anteriori sau o parte din ei, după care poate folosi resursele lor. Dacă nu există o astfel de secvenţă, starea respectivă de alocare este **nesigură**.

Metode de tratare a blocajelor
Prevenirea blocajelor

- sistemul trebuie proiectat în aşa fel încât să excludă posibilitatea blocării (este o soluţie foarte restrictivă ce limitează accesul la resurse şi impune restricţii proceselor).
- Prevenirea blocajelor se poate realiza prin:
  - înlăturarea excluderii mutuale (soluţie nerealistă);
  - înlăturarea stării de tip Hold-and-Wait :
    - la crearea proceselor să se realizeze cererile pentru toate resursele necesare;
    - blocarea proceselor până când toate cererile de acces pot fi satisfăcute simultan;
    - procesele să poată aştepta un timp îndelungat eliberarea resurselor;
    - resursele alocate unui proces ce pot rămâne un timp îndelungat nefolosite să poată fi utilizate de alte procese
  - înlăturarea planificării nepreemptive:
    - dacă unui proces îi sunt blocate unele cereri, atunci el să elibereze resursele alocate;
    - dacă un proces cere o resursă care este alocată altui proces, SO poate preempta al doilea proces şi să îi ceară să elibereze resursa;
  - înlăturarea aşteptării circulare:
    - definirea unei ordonări liniare a resurselor (o prioritate)
    - odată ce o resursă a fost obţinută, numai resursele care urmează din listă pot fi obţinute

Metode de tratare a blocajelor
Evitarea blocajelor

- Permiterea apariţiei a trei dintre condiţiile de blocare, dar trebuie asigurat faptul că nu se ajunge niciodată în starea de blocaj;
- O decizie este luată dinamic dacă alocarea curentă a resurselor poate duce la o situaţie de blocaj dacă este permisă – este necesară cunoaşterea posibilelor cereri ce pot apare.
- Se poate realiza dacă:
  - Nu se permite startarea unui proces dacă cererile de resurse pot duce la blocaj
  - Nu se permit cererile succesive de resurse ale unui proces dacă alocarea lor poate duce la blocaj

Metode de tratare a blocajelor
Detectarea blocajelor

- cererile de resurse sunt permise ori de câte ori este posibil; periodic, SO rulează algoritmi de detecţie a blocajelor.
- Strategii de rezolvarea situaţiilor în care se detectează blocaje:
  - sunt oprite toate procesele blocate;
  - se salvează starea proceselor blocate la un moment anterior apariţiei blocajului (checkpoint) şi se restartează procesele (este posibil să se ajungă din nou la blocaj)
  - procesele sunt oprite succesiv până când se iese din starea de blocaj
  - alocarea preemptivă a resurselor până la dispariţia blocajului
- Criterii de selecţie a proceselor blocate:
  - cel mai puţin timp procesor consumat până în prezent;
  - cele mai puţine rezultate produse până în prezent;
  - cel mai mult timp de rulare rămas estimat;
  - cele mai puţine resurse alocate din totalul celor cerute până în prezent;
  - cea mai mică prioritate.

Metode de tratare a blocajelor
Revenirea din blocaje

- oprirea tuturor proceselor blocate
- se salvează starea proceselor blocate la un moment anterior apariţiei blocajului (checkpoint) şi se restartează procesele:
  - se presupune că avem puncte de control a rulării şi mecanisme pentru restartarea proceselor din acele puncte de control;
  - este posibil să se ajungă din nou la blocaj (se presupune că dacă avem suficient timp, blocajul nu va reapare)
- procesele sunt oprite succesiv până când se iese din starea de blocaj
- alocarea preemptivă a resurselor până la dispariţia blocajului
- restartarea proceselor de la un punct anterior obţinerii resurselor
- achiziţionarea forţată a resurselor de la anumite procese şi alocarea lor altor procese până la eliminarea blocajului:
  - se alege un proces "victimă" (deadlock victim) de la care vor fi achiziţionate resursele, urmărindu-se asigurarea unui "cost" minim – ce poate include numărul resurselor ocupate şi mărimea duratei de timp de execuţie consumate deja de către acesta;
  - se reia execuţia procesului de la care a fost achiziţionată resursa (procesul este "întors în timp" pînă ajunge într-o stare sigură şi pornind de la această stare se reia execuţia);
  - "înfometarea": dacă în alegerea "victimei" sistemul se bazează în principal pe factorul cost, este posibil ca de fiecare dată să fie desemnat acelaşi proces ca "victimă", astfel că el nu va putea niciodată să-şi încheie normal execuţia. Pentru evitarea unor astfel de situaţii este necesar să se impună o limitare a numărului de alegeri ca "victimă", de exemplu prin includerea în cost a numărului de întoarceri în timp.

Metode de tratare a blocajelor
Alte soluții

- Gruparea resurselor în clase, cu o strategie de evitare a blocajului diferită pentru fiecare clasă de resurse:
  - **Memoria virtuală**: se previn situaţiile de blocaj prin impunerea alocării spaţiului de memorie virtuală necesar o singură dată;
  - **Sistemul de fişiere**: strategiile de evitare a blocajului pot fi implementate cu succes, prevenirea blocajului prin ordonarea resurselor este de asemenea posibil;
  - **Memoria centrală**: strategiile preemptive sunt foarte utile în acest caz;
  - **Resursele interne**: - resursele utilizate de către sistem (blocul de control al procesului)

Metode de tratare a blocajelor
Alte solutii (2)

- Ordonarea claselor de mai sus este folosind în cadrul fiecărei clase următoarele abordări:
  - prevenirea interblocării prin ordonarea resurselor interne (în timpul execuţiei nu este necesară alegerea uneia dintre cererile nerezolvate);
  - prevenirea interblocării prin achiziţie forţată a memoriei centrale (se poate evacua oricând un proces în memoria virtuală);
  - evitarea interblocării în cazul resurselor procesului (informaţiile necesare despre formularea cererilor de resurse pot fi obţinute din liniile de comandă)
  - alocarea prealabilă a spaţiului din memoria virtuală asociat fiecărui proces utilizator (în general se cunoaşte necesarul maxim de memorie al fiecărui proces).

Metode de tratare a blocajelor
Algoritmul bancherilor

- Fiecare nou proces apărut în sistem trebuie să declare numărul maxim de elemente din fiecare tip de resursă care i-ar putea fi necesar, număr ce nu poate depăşi numărul total de resurse din sistem.
- În momentul în care un proces formulează o cerere pentru un set de resurse, trebuie să se verifice dacă va lăsa sistemul într-o stare sigură, caz în care operaţia este permisă.
- Altfel, procesul trebuie să aştepte până când vor fi eliberate de către alte procese suficient de multe resurse care să satisfacă şi cererea sa.

Algoritmul bancherilor

- structuri de date pentru codificarea stării de alocare a resurselor sistemului:
- **m** – numărul maxim de instanţe pentru fiecare resursă (o resursă poate avea mai multe copii);
- **n** – numărul proceselor;
- **Disponibile[k];** – un vector de dimensiune **m** care indică numărul de resurse disponibile aparţinând fiecărui tip de resursă. Dacă **Disponibile[j] = k** înseamnă că din resursa **j** avem disponibile **k** copii;
- **Max[i][j]; i=1÷n, j=1÷m;** matrice de dimensiune **n x m** care indică numărul maxim de cereri ce pot fi formulate de către fiecare proces; dacă **Max[i][j] = k** înseamnă că procesul $\mathbf{p_i}$ poate cere cel mult **k** elemente de tip resursă $\mathbf{r_j}$;
- **Alocate[i][j]; i=1÷n, j=1÷m;** matrice de dimensiune **n x m** care indică numărul de resurse din fiecare tip care sunt alocate în mod curent fiecărui proces; dacă **Alocate[i][j] = k** înseamnă că procesul $\mathbf{p_i}$ are alocate **k** elemente de tip resursă $\mathbf{r_j}$;
- **Necesare[i][j]; i=1÷n, j=1÷m;** matrice de dimensiune **n x m** care indică numărul de resurse ce ar mai putea fi necesare fiecărui proces; dacă **Necesare[i][j] = k** înseamnă că procesul $\mathbf{p_i}$ ar mai avea nevoie în plus de **k** elemente din resursa $\mathbf{r_j}$ pentru a se termina; **Necesare =Max –Alocate**;

Algoritmul bancherilor
Algoritm de verificare a siguranței sistemului (de verificare a stării):

Temp[m];

Terminate[n];

Pas 1: *Temp =Disponibile; Terminate[i] = 0,* $\forall$ *i;*

Pas 2: Găseşte un *i* dacă există, astfel încât:

a) *Terminate[i] =0;*

b) *Temp* $\geq$ *Necesare[i];*

//linia din Necesare[i] corespunzătoare componentei *i*

Dacă nu există un astfel de i **goto Pas 4**.

Pas 3: *Temp =Temp + Alocate[i];*

//linia din Alocate[i][j] corespunzătoare componentei *i*

Terminate[i] =1; **goto Pas 2**.

Pas 4: Dacă *Terminate[i] =1*, $\forall$ *i*, atunci sistemul este într-o stare sigură.

Algoritmul bancherilor
Algoritm de verificare a cererii

- Dacă **Cererei[j] =k** , procesul $\mathbf{p_i}$ are nevoie de **k** elemente din tipul de resursă $\mathbf{r_j}$.
- Cînd $\mathbf{p_i}$ realizează o cerere de resurse, vor fi parcurse următoarele etape:

  Pas 1: Dacă $Cerere_i \leq Necesare_i$ **goto Pas 2**; altfel eroare (procesul a depăşit limita maxim admisă);

  Pas 2: Dacă $Cerere_i \leq Disponibile_i$ **goto Pas 3**; altfel procesul $\mathbf{p_i}$ este nevoit să aştepte (resursele nu sunt disponibile);

  Pas 3: Se simulează alocarea resurselor cerute de procesul $\mathbf{p_i}$ modificând starea de alocare a resurselor astfel:

  $$Disponibile_i = Disponibile_i - Cerere_i ;$$

  $$Alocate_i = Alocate_i + Cerere_i ;$$

  $$Necesare_i = Necesare_i - Cerere_i ;$$

- Dacă starea de alocare a resurselor rezultată este sigură, se alocă procesului $\mathbf{p_i}$ resursele cerute.
- Dacă noua stare este nesigură, procesul $\mathbf{p_i}$ trebuie să aştepte, iar sistemul reface starea de alocare a resurselor existentă înainte de execuţia **Pas 3**.

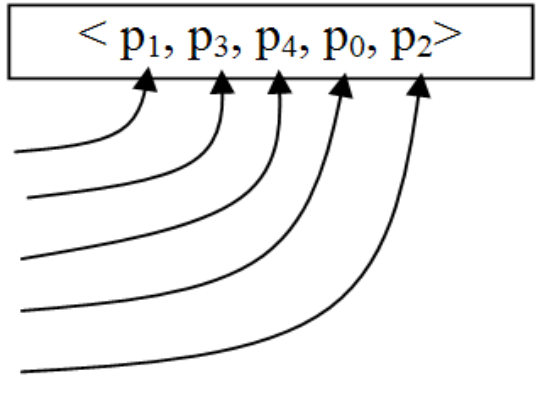
Algoritmul bancherilor
Exemplu:

- Avem 5 procese: p_0, p_1, p_2, p_3, p_4 şi 3 resurse A=10, B=5, C=7;

Nr. crt	Alocate	Max	Disponibile ABC	Necesare ABC
p_0	0 1 0	7 5 3		7 4 3
p_1	2 0 0	3 2 2		1 2 2
p_2	3 0 2	9 0 2	3 3 2	6 0 0
p_3	2 1 1	2 2 2		0 1 1
p_4	0 0 2	4 3 3		4 3 1

Algoritmul bancherilor Exemplu:

Trebuie să găsim o secvenţă de forma $< p_1, p_2, \ldots p_n >$



1) *Temp* = (3, 3, 2); *Terminate*[1] = 0; i=1, *Terminate*[1] = 1;
2) *Temp* = (5, 3, 2); *Terminate*[3] = 0; i=3, *Terminate*[3] = 1;
3) *Temp* = (7, 4, 3); *Terminate*[4] = 0; i=4, *Terminate*[4] = 1;
4) *Temp* = (7, 4, 5); *Terminate*[0] = 0; i=0, *Terminate*[0] = 1;
5) *Temp* = (7, 5, 5); *Terminate*[2] = 0; i=2, *Terminate*[2] = 1;

- Starea curentă este o stare sigură deoarece există o secvenţă de alocare (de satisfacere a necesarului) conform algoritmului verificare a siguranţei sistemului:
  - $< p_1, p_3, p_4, p_0, p_2>$
  - Condiţie: starea respectivă este sigură.
- Presupunem că $Cerere_1$ =(1,0,2);
- Se pune problema dacă cererea poate fi satisfăcută sau nu. Verificăm condiţiile de la paşii 1 şi 2 din algoritmul de verificare a cererii.

Algoritmul bancherilor
Exemplu:

	Alocate ABC	Disponibile ABC	Necesare ABC
p_0	0 1 0	2 3 0	7 4 3
p_1	3 0 2		0 2 0
p_2	3 0 2		6 0 0
p_3	2 1 1		0 1 1
p_4	2 0 2		4 3 1

- Daca se execută din nou algoritmul de verificare a siguranţei sistemului se găseşte secvenţa: $<p_1, p_3, p_4, p_0, p_2>$.
- Dacă în noua stare vom avea următoarele cereri:
  - $Cerere_4 = (3,3,0)$ sau $Cerere_0 = (0,2,0)$,
  - amândouă vor duce în stări nesigure şi nu vor putea fi satisfăcute (condiţiile din algoritmul de verificare a cererii vor fi satisfăcute numai pentru $Cerere_0$).

Algoritmul bancherilor
Algoritm pentru detecția blocajului

Pas 1: *Temp =Disponibile;*

for(i =1; i ≤ n , i++)

if(Alocate$_i$!= 0)

Terminate[i] =0;

else

Terminate[i] =1;

Pas 2: Găseşte *i* astfel încât:

a) *Temp ≥Cerere$_i$*

b) *Terminate[i] =0*

Dacă nu există *i* **goto Pas 4**.

Pas 3: *Temp =Temp +Alocate$_i$*

Terminate[i] =1;

goto Pas 2;

Pas 4: Dacă exista *i* astfel încât *Terminate[i] =0*, atunci starea respectiva este o stare de blocaj, deci procesul *i* este blocat.

Algoritmul bancherilor
Algoritm pentru detecția blocajului

- Exemplu:
  - procese şi resursele A =7; B=2; C =6

	Alocate ABC	Cerere ABC	Disponibile
p_0	0 1 0	0 0 0	0 0 0
p_1	2 0 0	2 0 2	
p_2	3 0 3	0 0 0	
p_3	2 1 1	1 0 0	
p_4	0 0 2	0 0 2	

Algoritmul bancherilor
Algoritm pentru detecția blocajului

- Executându-se algoritmul se gaseste secvenţa: <p_0, p_2, p_3, p_1, p_4> astfel încât *Terminate[i] =1*, $\forall$ *i* , deci starea respectivă nu este în blocaj.
- Presupunem că lucrăm cu o altă cerere $Cerere_2$ = (0,0,1)
- Rulându-se algoritmul observăm că starea respectivă este o stare de blocaj.
- Singurul proces care poate rula este p_0, în rest pentru toate celelalte procese vom avea *Terminate[i] =0*.

	Alocate ABC	Cerere ABC	Disponibile
p_0	0 1 0	0 0 0	0 0 0
p_1	2 0 0	2 0 2	
p_2	3 0 3	0 0 1	
p_3	2 1 1	1 0 0	
p_4	0 0 2	0 0 2	

Algoritmul bancherilor

- Cînd trebuie apelat algoritmul de detecţie?
  - Răspunsul depinde de doi factori:
    - cât de frecvent apare starea de blocaj
    - câte procese sunt afectate de blocaj.
  - Dacă blocajul apare frecvent, atunci algoritmul trebuie utilizat foarte des.
  - Resursele alocate proceselor blocate rămân iniţializate până se iese din starea de blocaj.