

# Sisteme de Operare

## Comunicații inter-procese și interblocajul proceselor

**Cristian Vidrașcu**

<https://profs.info.uaic.ro/~vidrascu>

- **Comunicații inter-procese (IPC)**
  - Problema comunicației
  - Tipuri de comunicație
  - Comunicația directă
  - Comunicația indirectă
  - Excepții
  - IPC sub UNIX
- **Interblocajul proceselor**
  - Definiție
  - Context
  - Model
  - Cerințele interblocajului
  - Graful de alocare a resurselor
  - Strategii: prevenire, evitare, detecție și restabilire
- **Înfometarea proceselor**
  - Definiție
  - Strategii de rezolvare

# Comunicații inter-procese

- Comunicații inter-procese  
(IPC = Inter-Process Communication)
- Problema comunicației
  - Proiectarea unor mecanisme care să permită **comunicația** între procesele ce doresc să coopereze; există 2 modele:
    - **Sisteme cu memorie partajată (modelul *shared memory*)** – necesită ca procesele comunicante să partajeze niște variabile (zone) de memorie
    - **Sisteme cu mesagerie (modelul *message-passing*)** – permit proceselor să schimbe mesaje între ele (chiar fără a avea memorie partajată)
  - Procesele ce comunică pot fi executate local sau la distanță

# Comunicații inter-procese

- Primitivele sistemelor cu mesaje:
  - send (*mesaj*)
  - receive (*mesaj*)
- Mesajele pot fi de lungime fixă sau de lungime variabilă
- Între procesele comunicante trebuie să existe o **legătură de comunicație**

# Comunicații inter-procese

- Întrebări legate de implementare:
  - Cum se stabilesc legăturile de comunicație între procese?
  - O legătură poate să fie asociată cu mai mult de două procese?
  - Câte legături pot exista între o pereche de procese?
  - Care este capacitatea unei legături?
  - Care este dimensiunea mesajelor?
  - O legătură este unidirecțională sau bidirecțională?

# Comunicații inter-procese

- Modalități folosite pentru implementarea fizică a legăturii de comunicație:
  - memorie partajată
  - magistrală hardware
  - rețea
- Modalități folosite pentru implementarea logică a legăturii de comunicație și a operațiilor `send()` și `receive()`:
  - comunicație directă sau indirectă
  - comunicație simetrică sau asimetrică
  - *buffering* (= stocarea mesajului într-o zonă tampon pentru preluarea ulterioară de către destinatar) implicit sau explicit
  - trimiterea mesajului prin copie sau prin referință
  - mesaje de lungime fixă sau de lungime variabilă

# Comunicații inter-procese

- Comunicație directă:
  - Primitivele `send()` și `receive()` specifică explicit numele procesului destinatar, respectiv expeditor (*comunicare simetrică*)
  - Legătura este stabilită automat între fiecare pereche de procese ce doresc să comunice; ele trebuie să cunoască doar identitatea celuilalt
  - O legătură este asociată cu exact două procese
  - Între fiecare pereche de procese comunicante există exact o legătură de comunicație
  - Legătura poate fi și unidirecțională, însă de obicei este bidirecțională

# Comunicații inter-procese

- Problema Producător-Consumator (reluare)
  - O soluție bazată pe comunicație directă
  - Procesul producător:

**repeat**

```
produ_element_in(nextpro) ;  
send(consumer, nextpro) ;
```

**forever**



# Comunicații inter-procese

- Problema Producător-Consumator (reluare)

- Procesul consumator:

- repeat**

- `receive(producer, nextcon) ;`

- `consuma_element_din(nextcon) ;`

- forever**

- Varianta: *comunicație asimetrică* – destinatarul poate primi de la oricine: `receive(id, mesaj)`
  - O altă soluție posibilă: bazată pe comunicație indirectă

# Comunicații inter-procese

- Comunicația indirectă
  - Mesajele sunt trimise/primate prin intermediul unor cutii poștale (numite și **porturi**)
  - Două procese pot comunica numai dacă au o cutie poștală în comun (i.e. partajată)
  - Primitivele de comunicație au forma:
    - **send(PORT,mesaj)**
    - **receive(PORT,mesaj)**

# Comunicații inter-procese

- Comunicația indirectă
  - Permite folosirea unei legături de comunicație de către mai multe procese
  - Permite la un moment dat cel mult unui proces să execute o operație `receive()` pe un anumit port
  - Permite sistemului să selecteze arbitrar care proces va primi mesajul (în caz de mai multe operații `receive()` efectuate “simultan” de procese distincte, acestea nu au nici un control asupra ordinii de satisfacere a cererilor lor)

# Comunicații inter-procese

- Comunicația indirectă
  - Operații asupra cutiilor poștale:
    - crearea unei cutii poștale
    - trimiterea/primirea de mesaje prin intermediul unei cutii poștale
    - distrugerea unei cutii poștale
  - Temă: reformulați soluția problemei producător-consumator folosind comunicația indirectă

# Comunicații inter-procese

- Comunicațiile pot fi blocante sau neblocante
  - Operații blocante (sau sincrone)
    - `send()` blocant: expeditorul e blocat până când mesajul este recepționat
    - `receive()` blocant: destinatarul este blocat până când un mesaj este disponibil pentru recepționare
  - Operații neblocante (sau asincrone):
    - `send()` neblocant: expeditorul trimite mesajul și-și continuă execuția
    - `receive()` neblocant: destinatarul recepționează imediat, fie un mesaj valid, fie mesajul null (ce semnifică faptul că încă nu s-a primit un mesaj)
  - Rezultă astfel 4 combinații posibile
    - combinația `send()` blocant + `receive()` blocant se numește *rendezvous*

# Comunicații inter-procese

- Excepții
  - Terminarea unui proces înainte de primirea mesajului
  - Pierderea mesajelor
  - “Amestecarea” (*scrambling*-ul) mesajelor
- Soluții
  - S.O.-ul trebuie să detecteze excepțiile, folosind diverse mecanisme:
    - notificări de terminare trimise celuilalt proces
    - *timeout*-uri și protocoale de confirmare
    - sume de control, de paritate, CRC, etc.

# Comunicații inter-procese

- IPC sub UNIX

- *pipe*-uri (canale anonime)
  - `pipe()`
  - pot fi utilizate doar de către procese înrudite prin `fork()`
- *named pipe*-uri (canale cu nume), i.e. fișiere `fifo`
  - `mkfifo()`
  - pot implementa schimbul de mesaje între procese oarecare, dar locale (i.e. executate pe același calculator)
- *socket*-uri (*Notă* : se vor studia la rețele de calculatoare, în anul 2)
  - pot implementa schimbul de mesaje între procese oarecare, la distanță (i.e. executate pe calculatoare diferite)
- *semnale* UNIX

# Comunicații inter-procese

- IPC sub UNIX (cont.)
  - semafoare (UNIX System V)
    - implementate cu mesaje
    - `semget()`, `semop()`, `semctl()` - UNIX System V
    - `creatsem()`, `opensem()`, `waitsem()`, `sigsem()` - Xenix
  - zone de memorie partajată (UNIX System V)
    - `shmget()`, `shmat()`, `shmdt()`, `shmctl()`
  - cozi de mesaje (UNIX System V)
    - `msgget()`, `msgsnd()`, `msgrcv()`, `msgctl()`
- Alte mecanisme de IPC (pentru UNIX și Windows)
  - RPC (Remote Procedure Calls) și RMI (RPC în Java)
  - standardul MPI (Message Passing Interface)



- Interblocajul (*deadlock*-ul) proceselor
  - Definiție
  - Context
  - Model
  - Cerințele interblocajului
  - Graful de alocare a resurselor
  - Strategii de rezolvare: prevenire, evitare, detecție și restabilire

## ➤ Definiție

- **Interblocaj**: două sau mai multe procese așteaptă la infinit producerea unor evenimente ce pot fi cauzate doar de unul sau mai multe dintre procesele ce așteaptă, prin urmare nici un eveniment nu se va produce niciodată.
- Există o “așteptare circulară” (necesară dar nu și suficientă pentru a avea interblocaj, după cum vom vedea puțin mai încolo).

## ➤ Context

- Problema interblocajului proceselor va fi studiată în contextul unui sistem de calcul abstract, în care procesele sunt în competiție pentru accesul la resurse (CPU, memorie, periferice I/O, etc.)

## ➤ Modelul matematic:

- Tipurile de resurse:  $R_1, R_2, R_3, \dots$   
e.g.  $R_1$ =procesor,  $R_2$ =imprimantă
- Fiecare resursă  $R_i$  are  $W_i$  instanțe (i.e. numărul de “unități” ale acelei resurse)  
e.g.  $R_2$  are  $W_2=2$  instanțe (adică sistemul respectiv are 2 imprimante)
- Procesele utilizează resursele astfel:
  - cerere de alocare a resursei necesare
  - utilizarea resursei pentru o perioadă finită de timp
  - eliberarea resursei

# Interblocaj – Cerințe

- Sunt *necesare* 4 condiții pentru a fi posibilă apariția interblocajului (Coffman, Elphic, Shoshani – 1971):
- 1. excluderea mutuală:** procesele solicită controlul în mod exclusiv asupra resurselor pe care le cer S.O.-ului
  - 2. hold & wait (păstrare & așteptare):** procesele păstrează resursele deja deținute în timp ce așteaptă să obțină alte resurse
  - 3. no preemption (ne-preemptie):** resursele deținute de un proces nu-i pot fi luate de către S.O. fără voia sa
  - 4. așteptare circulară:** se poate stabili o ordonare  $P_1, P_2, \dots, P_n$  a proceselor astfel încât  $P_1$  așteaptă o resursă deținută de  $P_2$ ,  $P_2$  așteaptă o resursă deținută de  $P_3$ , ... ș.a.m.d. ... ,  $P_n$  așteaptă o resursă deținută de  $P_1$

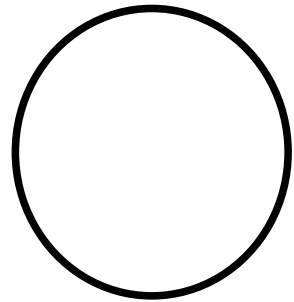
# Interblocaj – RAG /1

- Pentru modelarea sistemelor se folosește **graful de alocare a resurselor** (RAG = resource allocation graph), care ne arată:
  - Ce procese cer resurse și ce resurse cer ele
  - Ce resurse au fost acordate și căror procese au fost ele acordate
  - Câte unități din fiecare tip de resursă sunt disponibile

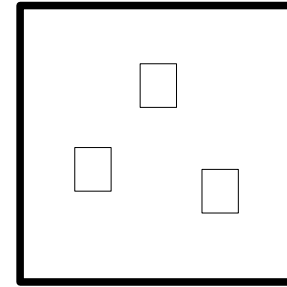
## Interblocaj – RAG /2

- Un RAG  $G$  este un graf bipartit orientat  $(V, E)$ , cu  $V$  mulțimea vârfurilor și  $E$  mulțimea arcelor
- Vârfurile din  $V$  sunt elemente de două tipuri:
  - $P = \{P_1, P_2, \dots, P_n\}$ : toate procesele din sistem
  - $R = \{R_1, R_2, \dots, R_m\}$ : toate resursele din sistem
- Arcele din  $E$  sunt de două tipuri:
  - arce de cerere: arc orientat  $P_i \rightarrow R_j$
  - arce de alocare: arc orientat  $R_j \rightarrow P_i$

# Interblocaj – RAG /3



Proces

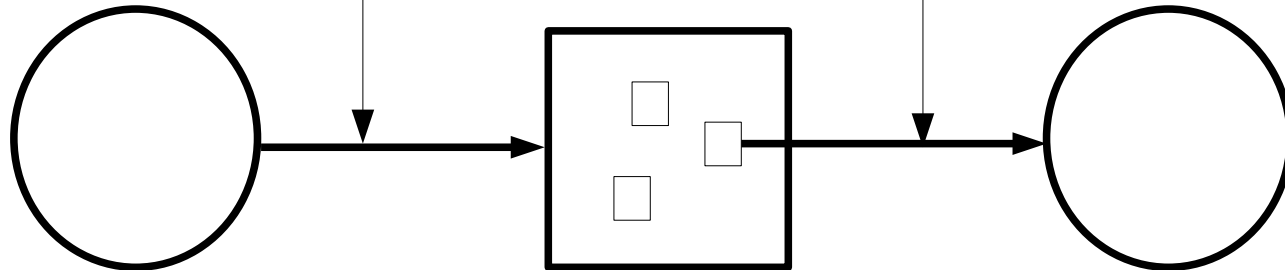


Resursă  
(3 unități)

Cerere  
și  
Alocare

Arc de cerere

Arc de alocare



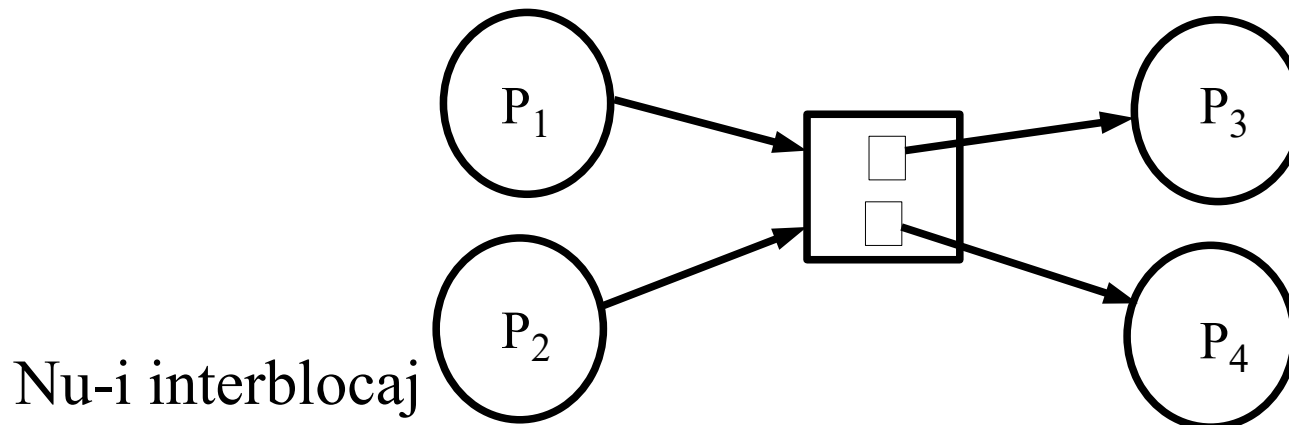
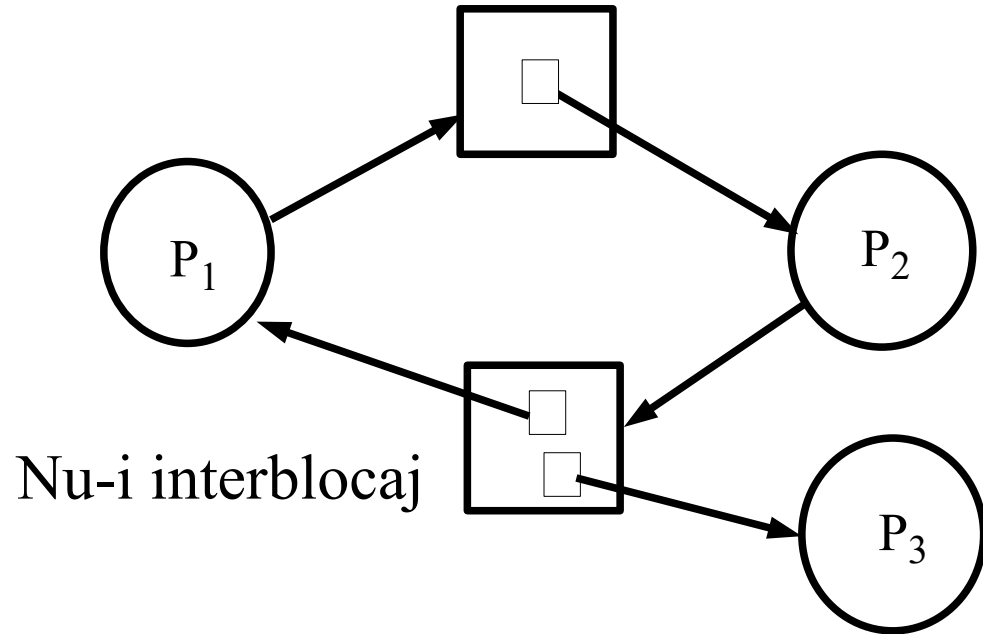
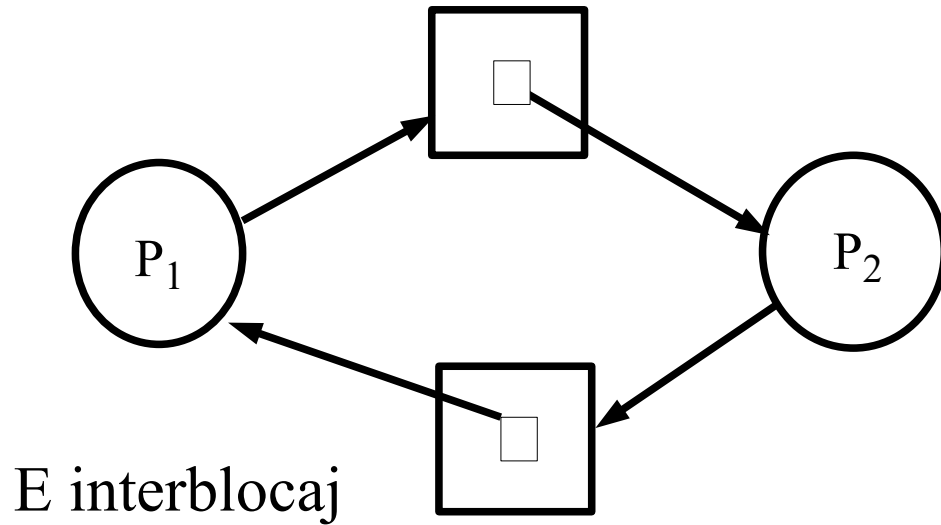


- Observații

- Dacă este interblocaj  $\rightarrow$  există circuit în RAG
- Dacă nu există circuite (orientate!) în graful de alocare a resurselor, atunci nu există interblocaj
- Dacă, în schimb, există un circuit în RAG, aceasta în general este o condiție necesară pentru interblocaj, dar *nu este și suficientă*  
(Observație: este și suficientă numai în cazul resurselor cu instanțe unice ; mai precis, doar resursele care intervin în circuit trebuie să fie cu instanțe unice)

# Interblocaj – RAG /5

Exemple



# Interblocaj – Strategii

## ➤ Strategii de rezolvare a interblocajului:

- **Ignorare**

- majoritatea S.O.-urilor ignoră interblocajele!

- **Prevenire**

- **Evitare**

➔ Ambele presupun utilizarea unui protocol care asigură faptul că sistemul nu va intra niciodată într-o stare de interblocaj

- **Detecție și restabilire**

➔ Se permite intrarea sistemului într-o stare de interblocaj, situație în care se face *recovery* (se iese din interblocaj)

# Interblocaj – Ignorare

## ➤ **Ignorarea interblocajului**

- Cea mai simplă metodă de rezolvare a interblocajului: nu-l rezolva!
- Te prefaci că nu va apare niciodată un interblocaj, sau presupui că dacă va apare, atunci sistemul se va comporta “ciudat”, moment la care va putea fi rebootat
- Avantaj: simplitate
- Dezavantaje:
  - Necesită intervenție inteligentă umană
  - Nesiguranța faptului dacă există într-adevăr interblocaj
  - S-ar putea să nu fie aplicabilă în sisteme reale critice

# Interblocaj – Prevenire /1

## ➤ **Prevenirea interblocajului**

- Presupune eliminarea a cel puțin uneia dintre cele 4 condiții necesare pentru interblocaj, făcând astfel interblocajul imposibil
- **Excluderea mutuală** → “relaxarea” (eliminarea) ei ar însemna: folosirea simultană, de către mai multe procese, a unei resurse
  - Evident, această condiție poate fi relaxată numai pentru resurse partajabile, nu și pentru resurse exclusive, e.g. imprimanta
- **Hold & wait** → “relaxarea” (eliminarea) ei ar însemna:
  - Nu lăsa procesul să păstreze resursele în timpul cât așteaptă să obțină alte resurse
  - Cum se poate implementa? De exemplu, procesul trebuie să obțină deodată toate resursele necesare (e.g., **prealocarea** = alocarea la începutul execuției a tuturor resurselor)
  - Dezavantaje: proasta utilizare a resurselor și posibilitatea de înfometare a proceselor

# Interblocaj – Prevenire /2

## ➤ **Prevenirea interblocajului (cont.)**

– **Ne-preemptie** → “relaxarea” (eliminarea) ei ar însemna:

- Preemptie, i.e. se permite S.O.-ului să ia înapoi resursele de la acele procese ce dețin resurse, dar așteaptă pentru alte resurse
- Resursele luate sunt adăugate la lista resurselor de care procesul “victimă” are nevoie pentru a-și continua execuția (ca și cum acel proces nu le-ar fi primit niciodată)
- Dezavantaj: pot apare complicații la luarea înapoi a resurselor. Soluția este aplicabilă doar pentru resurse pentru care contextul poate fi salvat și restaurat cu ușurință (e.g. CPU și memoria internă)

# Interblocaj – Prevenire /3

## ➤ Prevenirea interblocajului (cont.)

– **Așteptarea circulară** → “relaxarea” (eliminarea) ei ar putea fi:

- Se impune o ordonare a resurselor
  - discuri hard → 1
  - unitati CD-ROM → 2
  - imprimante → 3
- Procesele trebuie să ceară resursele în ordine crescătoare (sau înainte de a cere o resursă, trebuie să elibereze toate resursele deținute deja ce au numere de ordine mai mari), ceea ce elimină posibilitatea de așteptare circulară:

$P_1$ : request(disc);

request(imprimantă);

$P_2$ : request(imprimantă);

request(disc);

# Interblocaj – Evitare /1

## ➤ **Evitarea interblocajului**

- Strategia de evitare nu elimină vreuna din cele 4 condiții necesare pentru apariția interblocajului (ca la strategia de prevenire), ci în schimb folosește **alocarea controlată**: se examinează toate cererile de alocare a resurselor și se iau decizii astfel încât să se împiedice apariția interblocajului
- **Starea curentă** (i.e., la un moment dat) a sistemului: numărul și lista resurselor disponibile și respectiv alocate, precum și cerințele maxime de resurse ale proceselor
- **Stare sigură**: o stare în care sistemul poate alocă, într-o ordine oarecare, fiecărui proces alte resurse solicitate de către acesta, în limita maximului declarat, fără să apară un interblocaj



# Interblocaj – Evitare /2

## ➤ Evitarea interblocajului (cont.)

– Stare “nesigură” = *posibilitatea* de apariție a unui interblocaj, nu și *necesitatea* apariției lui

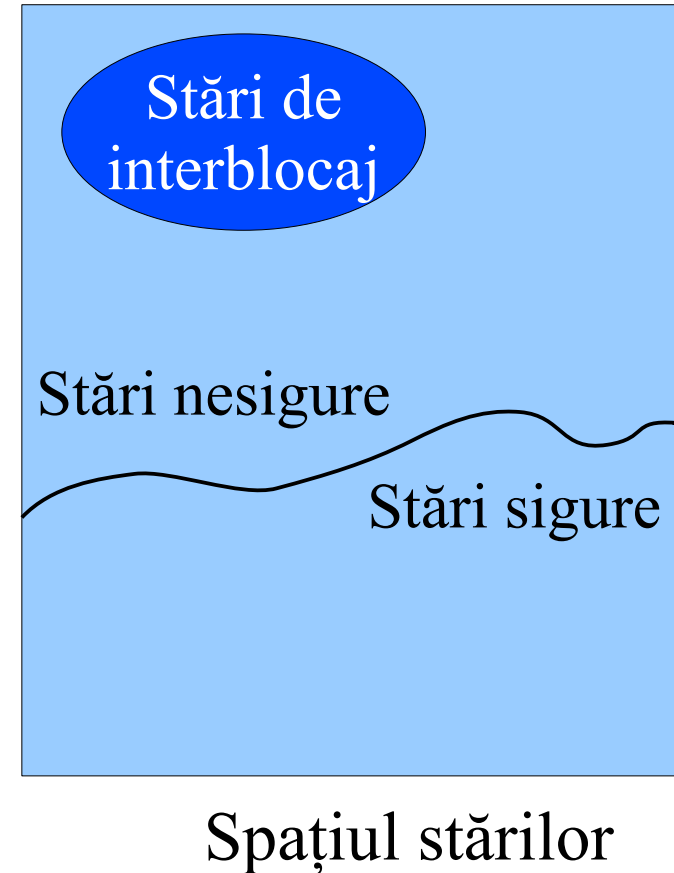
– Soluție:

- Un algoritm pentru păstrarea sistemului într-o stare sigură:

**algoritmul bancherului**

(the banker's algorithm)

[Dijkstra '65-'68, Habermann '69]



# Interblocaj – Evitare /3

- Algoritmul bancherului
  - Procesele trebuie să-și declare necesitățile maximale de resurse
  - Când un proces face o cerere de alocare de resurse, algoritmul bancherului determină dacă satisfacerea acestei cereri ar cauza intrarea sistemului într-o stare nesigură
    - Dacă da, atunci cererea este respinsă
    - Dacă nu, atunci cererea este aprobată

# Interblocaj – Evitare /4

- Algoritmul bancherului – structuri de date
  - $n$  = nr. proceselor ,  $m$  = nr. tipurilor de resurse
  - $\mathbf{Max}[1..n, 1..m]$  = cerința maximală din fiecare resursă a fiecărui proces

Dacă  $\mathbf{Max}[i,j] = k$ , atunci procesul  $P_i$  va solicita cel mult  $k$  instanțe din resursa  $R_j$  pe parcursul execuției sale
  - $\mathbf{Disponibil}[1..m](t)$  = nr. de instanțe disponibile curent (la momentul  $t$ ) în sistem, din fiecare resursă

Dacă  $\mathbf{Disponibil}[j] = k$ , atunci există  $k$  instanțe disponibile din resursa  $R_j$  la momentul respectiv

Inițial (la  $t=0$ ) :  $\mathbf{Disponibil}[j] = W_j$  (nr. de instanțe din  $R_j$ )

# Interblocaj – Evitare /5

- Algoritmul bancherului – structuri de date (cont.)
  - **Alocare**[1..n,1..m] (t) = alocarea curentă (i.e., la momentul t) a fiecărei resurse pentru fiecare proces  
Dacă **Alocare**[i,j] = k, atunci procesul  $P_i$  are alocate k instanțe din resursa  $R_j$  la momentul respectiv
  - **Necesar**[1..n,1..m] (t) = necesarul curent (i.e., la momentul t) din fiecare resursă a fiecărui proces  
Dacă **Necesar**[i,j] = k, atunci procesul  $P_i$  mai are nevoie de încă cel mult k instanțe din resursa  $R_j$  la momentul respectiv
  - Observații: **Necesar** (t) = **Max** – **Alocare** (t)  
și **Disponibil**[j] (t) =  $W_j - (\sum_{1 \leq i \leq n} \text{Alocare}[i,j] (t))$

# Interblocaj – Evitare /6

- Algoritmul bancherului – structuri de date (cont.)
  - **Cerere**[1..n,1..m] (t) = cererea curentă (i.e., la momentul t) din fiecare resursă a fiecărui proces  
Dacă **Cerere**[i,j] = k, atunci procesul  $P_i$  cere k instanțe din resursa  $R_j$  la momentul respectiv
  - Notatii:
    - **Cerere**<sub>i</sub> = vectorul cererii curente a procesului  $P_i$   
(i.e. **Cerere**<sub>i</sub> [j] (t) = **Cerere**[i,j] (t) )
    - **Necesar**<sub>i</sub> = vectorul necesarului curent al procesului  $P_i$   
(i.e. **Necesar**<sub>i</sub> [j] (t) = **Necesar**[i,j] (t) )
    - **Alocare**<sub>i</sub> = vectorul alocării curente a procesului  $P_i$   
(i.e. **Alocare**<sub>i</sub> [j] (t) = **Alocare**[i,j] (t) )

# Interblocaj – Evitare /7

- Algoritmul bancherului
  - Ideea: cunoașterea numărului maxim de instanțe din fiecare resursă pe care le poate cere un proces
  - Cunoașterea alocării de resurse curente a fiecărui proces și a necesarului curent (**max – alocarea curentă**)
  - Când apare o cerere, sistemul “pretinde” că o onorează ...
  - Și apoi, consideră cazul cel mai rău posibil: încearcă să satisfacă necesarul curent al tuturor proceselor într-o ordine oarecare ...
  - Dacă îi este imposibil, atunci refuză cererea

## Interblocaj – Evitare /8

- Algoritmul de soluționare a cererii – executat când este făcută o cerere de resurse de către un proces  $P_i$ 
  1. If  $Cerere_i > Necesari_i$  then { eroare; exit }  
// procesul și-a depășit cerința maximală pretinsă la început
  2. If  $Cerere_i > Disponibil$  then { wait( $P_i$ ); exit }  
//  $P_i$  trebuie să aștepte deoarece nu există resurse disponibile
  3.  $Disponibil := Disponibil - Cerere_i$ ;  
 $Alocare_i := Alocare_i + Cerere_i$ ;  
 $Necesari_i := Necesari_i - Cerere_i$ ;  
// sistemul “pretinde” că a alocat resursele solicitate de  $P_i$
  4. If  $EStareSigură()$  then { alocă resursele procesului  $P_i$  }  
else { wait( $P_i$ ); **rollback** pasul 3. } // refă vechea stare

# Interblocaj – Evitare /9

- Algoritmul de siguranță **EStareSigură()** – testează dacă starea curentă a sistemului este sigură sau nu
  1. Date: doi vectori **Work[1..m]**, **Finish[1..n]**  
Inițializări: **Work := Disponibil**, **Finish[i] := false**,  $i=1..n$
  2. Caută un  $i$  ce satisface condițiile  
**Finish[i] = false** and **Necesar<sub>i</sub> ≤ Work**  
// caut procesul  $P_i$  care ar putea fi servit din resursele disponibile
  3. If (există un astfel de  $i$ ) then  
{ **Work := Work + Alocare<sub>i</sub>**; **Finish[i] := true**; goto 2.; }  
// sper că  $P_i$  se va termina și voi putea folosi resursele lui  
else { goto 4.; }
  4. If (**Finish[i]=true for all i**) then { starea este sigură }  
else { starea este nesigură }



# Interblocaj – Evitare /10

## ➤ Algoritmul de siguranță **EStareSigură()** – exemplu

Considerăm un sistem cu un singur tip de resursă: unități de bandă magnetică, în număr de 12 unități, și cu 3 procese  $P_1$ ,  $P_2$  și  $P_3$ , care la un moment dat  $t$  au alocate respectiv câte 5, 2 și 2 unități de bandă, și care și-au declarat la început, ca și cerințe maxime, respectiv câte 10, 4 și 6 unități de bandă.

Deci:  $\text{Max} = (10, 4, 6)$ ,  $\text{Disponibil}(t) = 3$ ,

$\text{Alocare}(t) = (5, 2, 2)$  și  $\text{Necesar}(t) = (5, 2, 4)$

Este starea sistemului sigură? Da, este sigură, deoarece aplicând alg. de siguranță obținem o secvență sigură:  $\langle P_2, P_1, P_3 \rangle$  (de asemenea, și secvența  $\langle P_2, P_3, P_1 \rangle$  este sigură).

➤ Obs.: cu o mică modificare, alg. poate furniza și secvența sigură

# Interblocaj – Evitare /11

## ➤ Algoritmul de siguranță – al doilea exemplu

Un sistem cu 3 tipuri de resurse: A (10 instanțe), B (5 instanțe) și C (7 instanțe), având 5 procese  $P_0, P_1, \dots, P_4$ , care și-au declarat la început cerințele maxime și, la un moment dat  $t_0$ , au alocate resursele conform celor de mai jos:

	<u>Alocare</u> ( $t_0$ )	<u>Max</u>	<u>Disponibil</u> ( $t_0$ )	<u>Necesar</u> ( $t_0$ )
	<i>A B C</i>	<i>A B C</i>	<i>A B C</i>	<i>A B C</i>
$P_0$	0 1 0	7 5 3	3 3 2	7 4 3
$P_1$	2 0 0	3 2 2		1 2 2
$P_2$	3 0 2	9 0 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

Rezultă că vectorul Disponibil și matricea Necesar la momentul  $t_0$  sunt ca mai sus.

i) Este starea sistemului la  $t_0$  sigură? Da, este sigură, deoarece aplicând alg. de siguranță obținem o secvență sigură:  $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ .

ii) Procesul  $P_1$  face cererea (1 0 2) la  $t_0$ . Poate fi aceasta satisfăcută?

# Interblocaj – Evitare /12

➤ Algoritmul de siguranță – al doilea exemplu (cont.)

ii) Procesul  $P_1$  face cererea (1 0 2) la  $t_0$ . Poate fi aceasta satisfăcută?

Dacă ar fi satisfăcută, noua stare a sistemului la momentul următor  $t_1$ , ar fi:

	<u>Alocare</u> ( $t_1$ )	<u>Max</u>	<u>Disponibil</u> ( $t_1$ )	<u>Necesar</u> ( $t_1$ )
	<i>A B C</i>	<i>A B C</i>	<i>A B C</i>	<i>A B C</i>
$P_0$	0 1 0	7 5 3	2 3 0	7 4 3
$P_1$	3 0 2	3 2 2		0 2 0
$P_2$	3 0 2	9 0 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

Este starea sistemului la  $t_1$  sigură? Da, este sigură, deoarece aplicând alg. de siguranță obținem o secvență sigură:  $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ . Prin urmare, cererea procesului  $P_1$  la  $t_0$  poate fi satisfăcută.

iii) Procesul  $P_4$  face cererea (3 3 0) la  $t_0$ . Poate fi aceasta satisfăcută? ...

iv) Procesul  $P_0$  face cererea (0 2 0) la  $t_0$ . Poate fi ea satisfăcută? ...

# Interblocaj – Detecție /1

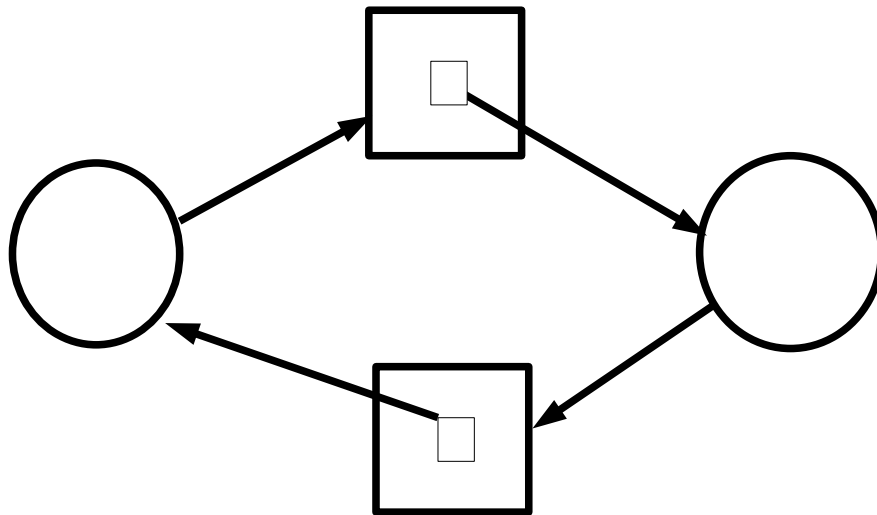
## ➤ Detecția interblocajului

- Pentru a detecta interblocajul, este nevoie de:
  - Un algoritm de detecție a interblocajului
  - Un mecanism de rezolvare (i.e., de a ieși din interblocaj)
  - Este mai ușor în cazul când toate resursele au instanțe unice, deoarece în acest caz un circuit în RAG este condiție necesară și suficientă pentru interblocaj
- Pentru a detecta interblocajul în cazul resurselor cu *instanțe unice*, graful de alocare a resurselor este “strâns” într-un graf de așteptare (**wait-for graph**) și se caută circuite în acest graf. Algoritmul de detecție a circuitelor are în acest caz complexitatea  $O(n^2)$ .

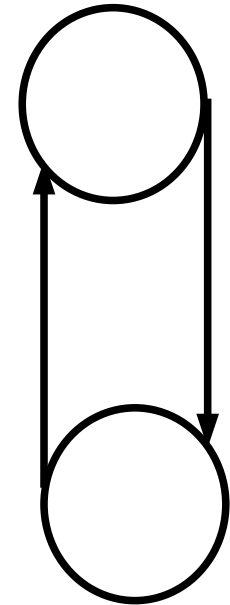
# Interblocaj – Detecție /2

## ➤ **Detecția interblocajului** (cont.)

- “Strângerea” unui RAG într-un graf wait-for:



devine:



- Observație: această tehnică nu funcționează în cazul resurselor cu instanțe multiple

# Interblocaj – Detecție /3

## ➤ **Detecția interblocajului** (cont.)

- Pentru cazul resurselor cu *instanțe multiple*, algoritmul de detecție a interblocajului operează în mod brut, încercând să reducă graful RAG prin verificarea tuturor posibilităților de alocare a resurselor
- Dacă nu există nici o posibilitate de alocare a instanțelor resurselor care să permită fiecărui proces să-și termine execuția, atunci este interblocaj
- Complexitatea algoritmului este în acest caz  $O(n^2 \cdot m)$ , cu  $n$  = numărul de procese și  $m$  = numărul de resurse  
Algoritmul este prezentat în continuare ...

# Interblocaj – Detecție /4

- Algoritmul de detecție – structuri de date
  - $n$  = nr. proceselor ,  $m$  = nr. tipurilor de resurse
  - **Disponibil**[1.. $m$ ] ( $t$ ) = nr. de instanțe disponibile curent (la momentul  $t$ ) în sistem, din fiecare resursă  
Dacă **Disponibil**[ $j$ ] =  $k$ , atunci există  $k$  instanțe disponibile din resursa  $R_j$  la momentul respectiv  
Inițial (la  $t=0$ ) : **Disponibil**[ $j$ ] =  $W_j$  (nr. de instanțe din  $R_j$ )
  - **Alocare**[1.. $n$ ,1.. $m$ ] ( $t$ ) = alocarea curentă (i.e., la momentul  $t$ ) a fiecărei resurse pentru fiecare proces  
Dacă **Alocare**[ $i,j$ ] =  $k$ , atunci procesul  $P_i$  are alocate  $k$  instanțe din resursa  $R_j$  la momentul respectiv

# Interblocaj – Detecție /5

- Algoritmul de detecție – structuri de date (cont.)
  - **Cerere**[1..n,1..m] (t) = cererea curentă (la momentul t) din fiecare resursă a fiecărui proces  
Dacă **Cerere**[i,j] = k, atunci procesul  $P_i$  cere k instanțe din resursa  $R_j$  la momentul respectiv
  - Notatii:
    - **Cerere<sub>i</sub>** = vectorul cererii curente a procesului  $P_i$   
(i.e. **Cerere<sub>i</sub>** [j] (t) = **Cerere**[i,j] (t) )
    - **Alocare<sub>i</sub>** = vectorul alocării curente a procesului  $P_i$   
(i.e. **Alocare<sub>i</sub>** [j] (t) = **Alocare**[i,j] (t) )
  - Obs.: **Disponibil**[j] (t) =  $W_j - (\sum_{1 \leq i \leq n} \text{Alocare}[i,j] (t))$



# Interblocaj – Detecție /6

➤ Algoritmul de detecție – testează dacă starea curentă a sistemului este interblocaj sau nu

1. Date: doi vectori  $Work[1..m]$ ,  $Finish[1..n]$

Inițializări:  $Work := Disponibil$ ,

$Finish[i] := (Alocare[i,j]=0 \text{ for all } j ? \text{true} : \text{false})$ ,  $i=1..n$

2. Caută un  $i$  a.î.  $Finish[i] = \text{false}$  and  $Cerere_i \leq Work$

// caut, printre procesele care dețin deja resurse, un proces  $P_i$  a cărui

// cerere ar putea fi servită din resursele disponibile

3. If (există un astfel de  $i$ ) then

{  $Work := Work + Alocare_i$ ;  $Finish[i] := \text{true}$ ; goto 2.; }

// sper că  $P_i$  se va termina și voi putea folosi resursele eliberate de el

else { goto 4.; }

4. If ( $Finish[i]=\text{true}$  for all  $i$ ) then { starea nu este interblocaj }  
else {este un interblocaj al proceselor  $i$  cu  $Finish[i]=\text{false}$ }

# Interblocaj – Detecție /7

## ➤ Algoritmul de detecție – exemplu

Considerăm un sistem cu un singur tip de resursă: unități de bandă magnetică, în număr de 12 unități, și cu 3 procese  $P_1, P_2$  și  $P_3$ , care la un moment dat  $t$  au alocate respectiv câte 5, 2 și 3 unități de bandă, și care mai solicită încă respectiv câte 5, 2 și 4 unități.

➔ Disponibil ( $t$ ) = 2, Alocare ( $t$ ) = (5,2,3) și Cerere ( $t$ ) = (5,2,4)

Este starea curentă a sistemului un interblocaj?

Nu, nu este, deoarece aplicând alg. de detecție obținem o secvență de servire a tuturor cererilor:  $\langle P_2, P_3, P_1 \rangle$ .

➤ Observație: cu o mică modificare, algoritmul poate furniza și secvența de servire.

# Interblocaj – Detecție /8

## ➤ Algoritmul de detecție – al doilea exemplu

Un sistem cu 3 tipuri de resurse: A (7 instanțe), B (2 instanțe) și C (6 instanțe), având 5 procese  $P_0, P_1, \dots, P_4$ , care, la un moment dat  $t_0$ , au alocate resurse și mai cer alte resurse conform celor de mai jos:

	<u>Alocare</u> ( $t_0$ )	<u>Cerere</u> ( $t_0$ )	<u>Disponibil</u> ( $t_0$ )	
	A B C	A B C	A B C	
$P_0$	0 1 0	0 0 0	0 0 0	Rezultă că vectorul Disponibil la momentul $t_0$ este ca în figură.
$P_1$	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		

i) Este starea sistemului la  $t_0$  un interblocaj? Nu, nu este interblocaj, deoarece aplicând alg. de detecție obținem o secvență de servire:  $\langle P_0, P_2, P_3, P_1, P_4 \rangle$ .

ii) Dacă la  $t_0$  cererea procesului  $P_2$  ar fi (0 0 1), în loc de (0 0 0), atunci sistemul ar fi în interblocaj la starea  $t_0$ , procesele interblocate fiind  $P_1, P_2, P_3$  și  $P_4$ .

# Interblocaj – Detecție /9

## ➤ **Detecția interblocajului** (cont.)

– Problemă:

- Cât de des ar trebui executat algoritmul de detecție?

– Răspunsul depinde de mai mulți factori:

- Cât de des este posibil să apară un interblocaj?
- Câte procese pot fi afectate de interblocaj?
- La apariția unui interblocaj, se poate întâmpla ceva foarte grav mai înainte ca un operator uman să observe ceva suspect în sistem?
- Cât de mult *overhead* al sistemului ne putem permite?

# Interblocaj – Restabilire /1

## ➤ **Ieșirea din interblocaj**

- Dacă s-a detectat un interblocaj, cum se poate restabili sistemul (i.e., ieși din blocaj) ?
- Răspunsuri posibile:
  - **Violarea excluderii mutuale** (numai pentru resurse partajabile, e.g. memorie, discuri hard, ...)
  - ***Abort*-ul** (i.e., terminarea anormală a) **tuturor proceselor**
  - ***Abort*-ul unui(or) proces(e) a.î. să dispară interblocajul** (Ce efect are *abort*-ul asupra integrității aplicațiilor?)
  - **Preemția unor resurse: selectarea unui(or) proces(e) victimă și luarea înapoi a resurselor deținute de el(e)** (Dezavantaje: necesitatea de a face *rollback* pentru victimă, posibilitatea de înfometare a victimei)

# Interblocaj – Restabilire /2

## ➤ **Ieșirea din interblocaj** (cont.)

- Câte procese să fie alese drept victimă (pentru abort sau pentru preempție & rollback) ?
  - Nr.minim = câte un proces pentru fiecare circuit disjunct în RAG
- Care procese? Răspunsul depinde de mai mulți factori:
  - Prioritatea proceselor
  - Cât timp a rulat procesul până la interblocaj și cât mai are de rulat până la terminarea sa
  - Câte resurse a utilizat procesul și ce fel de tip de resurse (ușor/difícil de preemptat)
  - De câte resurse mai are nevoie procesul pentru a se termina
  - Tipul procesului (interactiv sau serial)
  - Numărul de procese ce vor trebui abortate sau preemptate

# Interblocaj – Restabilire /3

## ➤ **Ieșirea din interblocaj (cont.)**

– Care procese să fie alese drept victimă (pentru abort sau pentru preempție & rollback) ? (cont.)

- Se definește o funcție de “cost” pe baza factorilor amintiți în slide-ul anterior, fiecare factor având o anumită pondere în cost
- Ponderile sunt alese conform preferințelor proiectanților SO-ului
- Alegerea victimei se determină prin minimizarea funcției de cost
- Posibilitatea de *starvation*: acel(e)ași proces(e) să fie ales(e) drept victimă întotdeauna, la execuția periodică, repetată, a alg. de detecție
- Rezolvarea acestui neajuns: includerea în funcția de cost a numărului de alegeri anterioare drept victimă, pentru fiecare proces

- Înfometare (*starvation*) a proceselor
  - Definiție
  - Strategii de rezolvare



## ➤ Definiție

- **Înfometare:** situația care apare atunci când un proces, ce a cerut permisiunea de acces la o resursă (e.g. la CPU, în cazul alg. de planificare a proceselor, sau la un semafor, în cazul alg. de sincronizare, etc.), așteaptă la infinit primirea acelei resurse, deoarece există un flux constant de alte procese care solicită acea resursă și o și primesc, în defavoarea procesului înfometat (datorită politicii de servire a acelei resurse implementate de acel S.O.).

## ➤ Strategii de rezolvare

- **Ignorare:** o lăsăm în sarcina operatorului uman
- **FIFO:** implementarea unei politici de servire care să respecte ordinea de primire a cererilor (e.g. la alocarea CPU-ului: alg. de planificare FCFS sau RR, sau pentru semafoare: implementarea cozii de așteptare sub formă de coadă FIFO, ș.a.).
- ***Aging-ul*:** implementarea unei politici de servire care să favorizeze procesele ce așteaptă de mult timp primirea resursei solicitate (e.g. la alocarea CPU-ului: alg. de planificare cu priorități dinamice, în care prioritatea unui proces este mărită treptat în timp ce este *ready*, cu revenirea la valoarea inițială a priorității după ce rulează o cuantă).

- **Bibliografie obligatorie**

capitolele despre *IPC* și despre *deadlock* din

- Silberschatz : “*Operating System Concepts*”

(cap.3,§3.4-8 despre IPC + cap.8 despre deadlock, din [OSC10])

sau

- Tanenbaum : “*Modern Operating Systems*”

(cap.2,§2.3.8-9 despre IPC + cap.6 despre deadlock, din [MOS4])

# Exercițiu de seminar

## ➤ Aplicație la: Interblocajul proceselor

### – Enunț:

Fie un sistem cu 4 procese  $P_i$ ,  $i=0,\dots,3$  și 5 tipuri de resurse alocabile, ce aplică o politică de evitare a interblocajului. La un moment dat  $t$  avem starea de mai jos.

- Completați tabelul cu conținutul matricii **Necesar**( $t$ ) din algoritmul bancherului.
- Care este cea mai mică valoare a lui "X" pentru care sistemul este în stare sigură la acel moment  $t$  ? Justificați răspunsul (specificați măcar o secvență sigură, dacă există).

	<b>Max</b>	<b>Alocare</b> ( $t$ )	<b>Disponibil</b> ( $t$ )	<b>Necesar</b> ( $t$ )
$P_0$	1 1 2 1 4	1 0 2 1 0	0 0 2 1 X	— — — — —
$P_1$	2 2 2 0 1	2 0 1 0 1		— — — — —
$P_2$	2 1 3 1 1	1 1 0 1 1		— — — — —
$P_3$	1 1 2 2 2	1 1 1 1 0		— — — — —

### – Rezolvare: ?

# Sumar

- **Comunicații inter-procese (IPC)**
  - Problema comunicației
  - Tipuri de comunicație
  - Comunicația directă
  - Comunicația indirectă
  - Excepții
  - IPC sub UNIX
- **Interblocajul proceselor**
  - Definiție
  - Context
  - Model
  - Cerințele interblocajului
  - Graful de alocare a resurselor
  - Strategii: prevenire, evitare, detecție și restabilire
- **Înfometarea proceselor**
  - Definiție
  - Strategii de rezolvare

# Gestiunea proceselor

## ➤ **Recapitulare**

- Definiția procesului
- Stările procesului
- Concurență
- Planificare
- Problema secțiunii critice
- Probleme clasice de sincronizare
- Comunicații inter-procese
- Interblocajul și înfometarea proceselor

## ➤ Urmează: **Administrarea memoriei**

~~Urmează: Primul parțial scris~~