# Sisteme de Operare 1 - Curs 11

cap 9 carte

pp. 245 - 270

*Curs tinut in* 2012-2013 *de catre* lector dr. Sanda-Maria Dragos

## Gesionarea proceselor

Facilitatea de a rula mai multe programe simultan in cadrul unui sistem de operare este considerata astazi normala de catre toti utilizatorii. Rularea unui navigator web simultan cu rularea unui program pentru citirea postei este o practica de zi cu zi a majoritatii utilizatorilor. Din punct de vedere a sistemului de operare toate aceste programe sunt considerate procese.

slide 2

**Program**

*- o secventa de instructiuni care descrie operatiile ce vor fi efectuate in timpul unei executii*

*- caracter static: nu se modifica de la executie la alta*

**Proces**

*- set de activitati din program care se executa secvential*

*- caracter dinamic: modul de desfasurare al prelucrarilor poate sa difere de la o executie la alta, in functie de contextul de lucru*

In arhitecturile monoprocesor, mai multe procese care ruleaza concomitent sunt de fapt deservite alternativ de catre procesor. Procesorul executa alternativ grupuri de instructiuni din fiecare program de-a lungul unei cuante de timp. Dimensiunea unei astfel de quante de timp este astfel aleasa incat momentele de stationare ale unui program nu sunt sesizabile de catre utilizator.

slide 3

Paralelismul este efectiv doar in cadrul sistemelor multiprocesor.

## Concurenta intre procese

Intr-un sistem pot coexista la un moment dat mai multe procese care solicita accesul concurent la resursele sistemului (memoria, discul magnetic, terminalul, interfata de retea, etc).

## Sectiune critica; resursa critica; excludere mutuala

O problema legata de accesul la resurse este asigurarea corectitudinii operatiilor executate in regim de concurenta.

slide 4

Pentru a vedea cum operatii corecte pot da rezultate gresite in caz de concurenta vom da un exemplu clasic. Se considera un fisier care stocheaza numarul de locuri libere intr-un sistem de vanzare de bilete. Vanzarea unui bilet va avea ca si consecinta decrementarea numarului de locuri libere. Codul care face acest lucru este:

```
int n;
int fd = open("locuri.db", "ORDWR");
read(fd, &n, sizeof(int));
lseek(fd, 0, SEEK_SET);
n--;
write(fd, &n, sizeof(int));
close(fd);
```

In cazul in care codul de mai sus este rulat simultan de mai multe procese diferite, este posibil ca instructiunile de mai sus sa se execute in ordinea din fig 1.

slide 5

Ambele procese citesc aceeasi valoare din program, o decrementeaza si apoi o scriu la loc. In final, in loc ca numarul de locuri sa scada cu doua pozitii, va scadea doar cu una si ca urmare operariile executate aspra fisierului sunt gresite.

Eroarea apare din lipsa de sincronizare intre accesele proceselor la fisier. Pentru a obtine rezultate corecte, secventa de executie dorita este:

slide 6

Portiunea de program incepand cu *read* si terminand cu *write* este o **sectiune critica** (deoarece NU este permis ca ea sa fie executata de mai multe procese simultan; adica sectiunea critica este neinteruptibila), iar pozitia pe care se stocheaza numarul de locuri in fisier este o **resursa critica** (deoarece nu poate fi accesata simultan de mai multe procese).

Pentru a evita situatia eronata prezentata anterior, vom spune ca procesele A si B trebuie sa se **excluda reciproc**, deoarece trebuie sa aiba access exclusiv la sectiunea si la resursa critica.

**Sincronizarea executie proceselor: Conceptul de semafor**

Dijkstra a introdus, pe la sarsitul anilor '60, conceptul de semafor ca si solutie la problema sectiunii critice (ca un fel de blocare generalizata). Un **semafor** este o pereche *(v(s), c(s))* , unde v(s) este valoare semaforului, iar c(s) o coada de asteptare. Valoarea v(s) este un numar intreg, iar c(s) contine pointeri la procesele care asteapta la semaforul *s*. Disciplina cozii depinde de sistemul de operare (FIFO, LIFO, prioritati, etc).

slide 7

Pentru gestiunea semafoarelor se definesc doua operatii indivizibile *P(s)* si *V(s)* ale caror roluri sunt "a trece de resursa", respectiv "a anunta eliberarea resursei" (Literele - primele din denumirea in olandeza a operatiilor). In alte resurse operatie *P* se numeste *WAIT*, iar operatie *V* se numeste *SIGNAL*. Definitiile exacte ale operatiilor *P* si *V* sunt date in fig

Folosirea *semaforului* in vederea solutionarii sectiunii critice se face sub forma: valoare initiala a semaforului se seteaza la 1: v0(s)=1, iar toate procesele care folosesc sectiune critica sunt de forma:

```
...; P(s); < sectuinea critica >; V(s); ...
```

Se poate demonstra ca v(s)<1, indiferent de numarul proceselor care opereaza asupra lui s cu operatiile *P* sau *V*, ceea ce inseamna ca un singur proces poate trece de semafor.

**Sincronizarea executie proceselor, exemple: Problema producatorului si a consumatorului**

Se presupune ca exista unul sau mai multe procese numite *producatoare*, si unul sau mai multe procese numite *consumatoare* (conceptele de pipe si FIFO sunt de aceasta natura). Transmiterea informatiilor de la producatori spre consumatori se realizeaza prin intermediul unui buffer cu *n* intrari pentru *n* articole. Fiecare producator depune cate un articol in buffer, iar fiecare consumator scoate cate un articol din buffer. Problema consta in a dirija cele doua tipuri de procese astfel incat:

slide 8

» sa existe acces exclusiv la buffer
» consumatorii sa astepte cand bufferul este gol
» producatorii sa astepte cand bufferul este plin

Rezolvarea prin semafoare este foarte simpla.
Se folosesc 3 semafoare: *gol* (nr. pozitii libere din buffer), *plin* (nr. pozitii ocupate din buffer) si *exclus*.

***Intrebari***: *Conteaza ordinea semafoarelor?* (DA)
*Conteaza ordinea operatiilor P?* (DA) *Conteaza ordinea operatiilor V?* (NU) ; ***DA = impas***

## Problema impasului

Impasul este un fenomen intalnit in viata de zi cu zi. El poate fi definit ca o blocare a activitatii normale a doua sau mai multe entitati ca urmare a efectului lor coroborat asupra mediului. Un exemplu este prezentat in fig de mai jos unde traficul din jurul cvartal-ului nu mai poate continua fara ca una dintre masini sa dea inapoi.

In domeniu informatic, o modalitate usoara de a crea o situatie de impas este inversarea operatiilor *P* si *V* in procese concurente.

Procesul A va detine semaforul x, iar procesul B va detine semaforul y. Ambele vor incerca sa obtina semaforul detinut de celalalt, fara a elibera semaforul pe care deja il detine inainte de a-l obtine pe cel detinut de celalalt proces.

Acest fenomen mai este cunoscut in literatura de specilitate sub mai multe denumiri: *impas*, *interblocare*, *deadlock*, *deadlock embrace*, etc.

In cazul in care doua procese se blocheaza unul pe celalalt, se poate iesi din impas terminand fortat unul din ele. In cazul in care impasul se manifesta intre un proces si sistemul de operare, sistemul ingheata, iar iesirea din astfel de situatii este restartarea calculatorului!

Fiecare sistem de operare trebuie sa rezolve macar una dintre urmatoarele probleme:

» iesirea din impas
» detectarea impasului
» evitarea (prevenirea aparitiei) impasului

slide 9

slide 10

**Iesirea din impas**

Pentru iesirea din impas solutiile pot fi *manuale* sau *automate*. Iesirea automata din starea de impas persupune in primul rand detectarea acestei stari, ceea ce implica existenta unei componente a sistemului de operare care sa intretina si sa analizeze starea alocarilor de resurse din sistem.

Solutiile iesirii din impas necesita actiuni drastice:

**Reincarcarea sistemului de operare**

*Solutie nedorita, insa daca pierderile nu sunt prea mari se poate adopta o astfel de strategie.*

**Alegerea unui "proces victima"**

*Acest proces este fie cel care a provocat impasul, fie un altul de importanta mai mica, astfel incat sa fie inlaturat impasul.*

**Creearea unui "punct de reluare"**

*Acest punct de reluare este o fotografie a memoriei pentru procesul victima si pentru procesele cu care el colaboreaza.*

**Detectarea impasului**

Detectarea impasului se face atunci cand SO nu are un mecanism de prevenire a impasului. E necesara o evidenta clara, pentru fiecare proces, privind resursele ocupate, precum si cele solicitate dar neprimite. Cel mai potrivit model este *graful alocarii resurselor*.

slide 11

Notam fiecare resursa: *R1, R2, ..., Rm*. Fie *P1, P2, ..., Pn* procesele din sistem. Se construieste un graf bipartit (X, U) astfel:

» X = {P1, P2, ..., Pn, R1, R2, ..., Rm}
» (Rj, Pi) este un arc in U daca procesul Pi a ocupat resursa Rj
» (Pi, Rj) este un arc in U daca procesul Pi asteapta sa ocupe resursa Rj

daca graful (X, U) definit mai sus este ciclic, atunci sistemul se afla in stare de impas.

**Evitarea (prevenirea aparitiei) impasului**

S-au determinat 4 conditii necesare pentru aparitia impasului.

slide 12

**Conditia de excludere mutuala**
*Procesele solicita controlul exclusiv asupra resurselor pe care le cer.*

**Conditia de ocupa si asteapta**
*procesele pastreaza resursele deja ocupate atunci cand asteapta alocarea altor resurse*

**Conditia de nepreemptie**
*Resursele nu pot fi eliberate din procesele care le tin ocupate, pana ele nu sunt utilizate complet. (Nici unui proces nu ii pot fi luate in mod fortat resursele.)*

**Conditia de asteptare circulara**
*Exista un lant de procese in care fiecare dintre ele asteapta dupa o resursa ocupata de altul din lant*

Evitarea impasului presupune impiedicarea aparitie uneia dintre aceste conditii.

1. **Conditia de excludere mutuala**
   - excluderea mutuala este esentiala pentru pastrarea corectitudinii datelor, deci nu se

poate elimina!

2. **Conditia de ocupa si asteapta**
   - *regula 1*: orice proces trebuie sa elibereze toate resursele blocate inainte de a solicita o noua resursa
   - *regula 2*: orice proces trebuie sa blocheze toate resursele de care are nevoie la pornire
3. **Conditia de nepreemptie** - oprirea unui proces nu garanteaza iesirea din impas a celorlalte procese pana in momentul cand acel proces (repornit) va reajunge in regiunea critica. Aceasta secventa se poate repeta la nesfarsit si desi nici un proces nu e blocat, totusi niciunul nu progreseaza cu executia. Acest fenomen este cunoscut sub numele de *impas activ* sau *livelock*.
4. **Conditia de asteptare circulara**
   - *regula*: impunerea unei oridini de blocare a resurselor, astfel incat nici un proces nu are voie sa blocheze o resursa cu numar de ordine mai mare inaintea uneia cu numar de ordin mai mic. Pentru a observa efectul blocarii ordonate prezentam *problema filozofilor* (problema de sincronizare propusa de Dijkstra in 1965). Se considera o masa circulara la care sunt asezati 5 filozofi. Fiecare filozof are perioade alternative de gandire si hranire. Pe masa, in loc sa fie 10 furculite sunt doar 5. Fiecare filozof are nevoie de doua furculite pentru a manca, si odata ce ia tacamul de pe masa nu il elibereaza pana nu termina de mancat. Daca toti filozofii iau tacamul din stanga ne aflam intr-o situatie de impas, pentru ca nici un filozof nu va avea 2 tacamuri pentru a manca.

   Regula de folosire care se impune este: daca se numeroteaza fiecare furculita, atunci fiecare filozof ia furculite in ordine crescatoare, eliberandu-le in ordine inversa (descrescatoare).
   Avand in vedere cantitatea mare de resurse dintr-un sistem, respectarea unei ordini prestabilite e imposibila. In plus, ordirea resurselor difera in cele mai multe cazuri de ordinea logica in care un program are nevoie de resurse.

**Alocarea controlata de resurse**

O alta modalitate de evitare a impasului este prin alocarea restrictiva a resurselor. Ceea ce implica ca fiecare cerere de resurse se va servi doar daca SO decide ca servirea cererii nu va genera o stare de impas.

**Algoritmul bancherului (Dijkstra)**

*Bancherul (SO) decide cand si cat (resurse) sa imprumute unui client (procesul) dintr-o suma maxima cu care il poate credita. In conditiile in care bancherul dispune de o suma limitata de imprumut; el nu percepe dobanda si comisioane; toti clientii returneaza banii in cele din urma.*

# Conceptul de multiprogramare

Modul de exploatare a unui sistem de calcul cu un singur procesor central, care presupune existenta simultana in memoria interna a mai multor procese care se executa concurent.

slide 13

In fiecare moment procesorul executa o instructiune a unui proces. Acest proces se afla in starea RUN. Celelalte procese din memorie pot fi in una din starile: SLEEP daca asteapta aparitia unui eveniment extern, sau READY daca sunt pregatite de executie.

Trecerea unui proces din starea RUN in starea SLEEP este comandata de catre procesor, cand intalneste o operatie I/O. Trecerea proceselor din starea RUN in starea READY poate fi comandata de proces (*cedare voluntara de procesor* - metoda veche; nu se mai practica) sau de catre SO (*cedare involuntara de procesor*).

Trecerea unui proces dintr-o stare in alta este facuta pe baza unui *algoritm de planificare* rulat de catre o componenta a SO numita *planificator*.

# Planificarea proceselor

slide 14

## Functionarea planificatorului de procese

Executarea a 3 procese cu prioritati diferite (P1 cel mai prioritar, P3 cel mai putin prioritar).

P3 reuseste sa intre in starea RUN doar cand P1 sai P2 sunt in starea SLEEP sau FINISH.

## Sarcinile planificatorului de procese

Planificatorul de procese este responsabil pentru trecerea proceselor din starea READY in starea RUN si invers. Pentru aceasta planificatorul indeplineste urmatoarele sarcini:

» Tinerea evidentei tuturor proceselor din sistem;
» Alegerea procesului caruia i se va atribui procesorul si pentru cat timp;
» Alocarea procesorului unui proces;
» Eliberarea procesorului la iesirea procesului din starea RUN.

slide 15

slide 16

Planificatorul mai intretine:

» Cate o structura de date pentru fiecare proces din sistem. Aceasta structura se numeste *Process Control Block (PCB), si contine:*
*» starea procesului*
*» pointer spre urmatorul PCB cu aceeasi stare*
*» numarul procesului*
*» contorul de proces (adresa instructiunii masina ce urmeaza a fi executata);*
*» zona pentru copia registrilor generali ai masinii*
*» limitele zonelor de mem. alocate procesului*
*» lista fisierelor deschise*
*» cozi la perifericele de I/O.*

## *Algoritmi de planificare a proceselor*

### *FCFS (First Come First Served)*

*Procese sunt servite in ordinea lor cronologica. De ex. pentru 3 procese care au timpii de executie: 24, 3, 3 minute,*
*daca vin in ordinea 1, 2, 3 timpul mediu de servire va fi (24+27+30)/3 = 27 minute.*
*Daca vin in ordinea 2, 3, 1, timpul lor mediu de servire va fi: (3+6+30)/3=13 minute*

*Algoritm simplu dar nu foarte eficient.*

*slide 17*

***SJF (Shortest Job First)***

*Executa primul procesul cu timpul cel mai scurt de executie.*

*Dezavantaj: trebuie cunoscuti timpii necesari de executie ai proceselor.*

***Algoritm bazat pe prioritati***

*Cel mai de folosit.*

*Exista 2 metode de stabilire a prioritatilor:*

1. *Procesul primeste prioritatea la intrarea in sistem si o pastreaza pana la sf.*
2. *SO calculeaza prioritati dupa reguli proprii si le ataseaza, dinamic, proceselor in executie*

   *» Dezavantaj: Procesele cu prioritate mica pot astepta indefinit ... fenomen numit starvation. Se poate folosi o metoda combinata prin care la prosele cu prioritate mica li se va mari prioritatea cu cat asteapta mai mult si se readuce la val initiala dupa ce intra in executie.*

***Algoritm bazat pe termene de realizare (deadline scheduling)***

*Destinat SO cu cerinte de timp real foarte strict. Fiecarui task i se ataseaza un termen de terminare. Planificatorul foloseste aceste termene pentru a decide carui proces ii va aloca procesorul pentru ca toate procesele sa termine la timp.*

***Round-Robin (planificare cirulara)***

*Destinat SO care lucreaza in timesharing. Se defineste o cuanta de timp (intre 10-100 milisecunde). Coada READY e tratata circular, ficarui proces alocandu-se procesorului pe durata unei cuante de timp, dupa care acel proces trece la sf. coadei.*

***Algoritmul de cozi pe mai multe nivele***

*Se aplica cand lucrarile pot fi usor clasificate in grupe distincte.*