ALGORITMI DE PLANIFICARE A EXECUȚIEI PROCESELOR

Radu Constantinescu

Academia de Studii Economice, Bucureşti

Rezumat: Metodele matematice pentru planificarea execuției proceselor reprezintă nucleul activității de planificare. De la caz la caz, pot fi propuse diverse metode matematice de planificare, pentru a satisface cât mai bine cerințele de bază, pentru funcționarea sistemului, referitoare la executarea cât mai rapidă a sarcinilor în condiții de stabilitate. Dintre metodele matematice analizate, amintim planificarea în funcție de priorități, planificarea round robin, planificarea SFJ sau planificarea FIFO.

Cuvinte cheie: proces, planificare, round-robin, FIFO, SFJ.

1. Introducere

Un proces reprezintă un set de activități din program, care nu se pot suprapune (se execută secvențial). Un proces este determinat de o secvență de instrucțiuni (o procedură) din programul care se execută. Termenul de proces este folosit pentru a desemna obiectul activității procesorului, în timpul execuției unui program.

Algoritmii de planificare a execuției proceselor sunt împărțiți, în principiu, în două mari clase: nonpreemptivi şi preemptivi. Algoritmii nonpreemptivi sunt proiectați ca, atunci când un proces intră în execuție, să fie rulat până când a fost epuizat timpul său de execuție. Algoritmii preemptivi se conduc după noțiunea de prioritate; procesul cu prioritatea cea mai mare trebuie să fie cel care foloseşte procesorul. Dacă un proces foloseşte procesorul la momentul curent şi un nou proces cu o prioritate mai mare intră în lista de procese gata de execuție, atunci procesul ce foloseşte procesorul trebuie pus în aşteptare în listă, până când devine din nou procesul cu cea mai mare prioritate din sistem. Algoritmii preemptivi sunt, în mod normal, asociați cu sistemele care folosesc întreruperi, pentru a forța împărțirea involuntară a accesului la procesor, în timp ce algoritmii nonpreemptivi sunt asociați cu sistemele bazate pe împărțirea voluntară a accesului la procesor.

2. Strategii de planificare a execuției proceselor

Vom prezenta succint principalele strategii de planificare cunoscute, specificând caracteristicile principale şi ilustrând grafic modul de lucru al acestora.

A. Primul venit - Primul servit (FIFO)

Strategia de planificare a proceselor *Primul Venit – Primul Servit* ataşează proceselor priorități în ordinea cărora acestea vor avea acces la unitatea centrală de prelucrare. Aceste priorități sunt calculate de către administratorul cozii de aşteptare la inserarea procesului, prin atribuirea unei mărci de timp tuturor proceselor. Apoi, dispecerul selectează procesul cu cea mai veche marcă de timp pentru execuție. Astfel, lista proceselor gata de execuție poate fi organizată ca o simplă structură de date *FIFO* (unde fiecare intrare punctează către un descriptor de proces). Administratorul cozii de aşteptare adaugă procesele la sfârşitul cozii, iar dispecerul extrage procesele din capul cozii. Strategia *FIFO* este pur nonpreemptivă.

Dacă algoritmii *FIFO* sunt simplu de implementat, aceştia ignoră cererile de execuție şi toate celelalte criterii care pot influența performanța, ținând seama doar de timpul de revenire sau de aşteptare. În general, *FIFO* nu se comportă bine sub nici un set specific de cereri ale sistemului, de aceea el nu este des folosit.

B. Cel mai scurt job rulează primul

Presupunem că timpul de servire al tuturor proceselor este cunoscut apriori. Algoritmul „*Cel mai scurt job rulează primul*" (*SJN* - Shortest Job Next sau *SJF*-Shortest Job First) alege pentru execuție procesul care necesită timpul minim de servire la următoarea intrare. Timpul de reîntoarcere pentru p_i este suma tuturor timpilor de servire ai proceselor din lista de aşteptare, care au timpi de servire mai mici ca p_i.

SJN minimizează timpul mediu de aşteptare deoarece el serveşte procese scurte, înainte de servirea celor ce necesită un timp lung de rulare. Cât timp acesta minimizează timpul mediu de aşteptare, poate penaliza procesele care necesită timp mare de servire. Dacă lista de aşteptare este saturată, procesele cu timp mare de aşteptare ajung în coada listei de planificare, în timp ce procesele cu timp mic de aşteptare sunt servite imediat. În cazul extrem, când sistemul are puțin timp nefolosit, procesele cu timp mare de servire nu vor fi niciodată servite.

Strategia poate fi implementată atât sub forma preemptivă, cât şi sub cea nepreemptivă. În cazul strategiei preemptive, dacă la un moment dat se execută procesul p_i şi un alt proces p_j este gata de execuţie, atunci algoritmul *SJN* trebuie doar să compare $T(p_i)$ cu $T(p_j)$ deoarece este garantat că p_i are cel mai mic timp de execuţie, dintre toate procesele gata de execuţie de la momentul când apare p_j.

C. Planificarea pe bază de priorităţi

În planificarea cu priorităţi, procesele sunt alocate procesorului, pe baza unei priorităţi externe asociate (în exemplul următor numerele mici au prioritate mai mare, unele planificatoare folosesc ordinea inversă). Prioritaţile interne derivă din modul în care planificatorul atribuie unitatea centrală de prelucrare a proceselor, cum ar fi, de exemplu, prioritatea determinată de lungimea timpului de servire, folosit de algoritmul *SJN*. Prioritaţile externe reflectă importanţa sarcinii pe care procesul trebuie să o îndeplinească. Prioritatea externă a unui proces poate fi dedusă fie din identitatea utilizatorului (persoanele importante au prioritate mare), fie din natura sarcinii (un program ce afişează un film va avea prioritate mai mare decât un program ce tipăreşte un document la imprimantă).

Dacă la un moment anume se execută procesul p_i şi apare un proces p_j cu prioritate mai mare, acest lucru va cauza reîntoarcerea lui p_i în lista proceselor gata de execuţie, în timp ce procesorul va fi alocat noului proces.

O cheie a performanţei unei planificări pe bază de priorităţi este alegerea priorităţii proceselor. Din nou, planificarea cu priorităţi poate cauza „înfometarea" proceselor cu prioritate joasă. Această „înfometare" poate fi compensată de calcularea internă a prioritaţilor. Să presupunem că un parametru din funcţia de atribuire a priorităţii unui proces este durata de timp în care procesul a aşteptat accesul la unitatea centrală de prelucrare. Cu cât un proces aşteaptă mai mult, cu atât prioritatea sa devine mai mare. Această strategie tinde spre a elimina problema „înfometării".

D. Planificare cu momente critice finale

Sistemele de timp real sunt, adesea, caracterizate ca sisteme ce au anumite procese a căror execuţie trebuie terminată înaintea unei limite de timp. Măsura performanţei acestor sisteme este dată de capacitatea de a respecta toate aceste limite de timp ale proceselor; măsurarea timpilor de reîntoarcere şi de aşteptare fiind irelevante, în general. Ca rezultat, aceste planificatoare trebuie să aibă cunoştinţe complete asupra duratei de execuţie a unui proces. Un proces poate fi admis în lista de aşteptare numai dacă planificatorul poate garanta că va fi în stare să se încadreze în limitele de timp impuse de acesta.

În sistemele multimedia, momentele critice de timp final pot fi cerute pentru a preveni transmiterea neregulată de informaţie şi decalajul între procesele audio şi video. În calculatoarele de proces (controlează un proces extern), timpul critic final poate fi stabilit de un senzor extern.

E. Planificare circulară (Round-Robin)

Algoritmul de planificare round-robin este probabil cel mai folosit dintre toţi algoritmii de planificare a proceselor. În esenţă, algoritmul round-robin distribuie echitabil timpul de procesare între toate procesele care cer acces la procesor. Această distribuţie va tinde să se adapteze filosofiei multiprogramării interactive, în care fiecare dintre cele *n* procese primesc aproximativ *1/n* unităţi de timp, din timpul de procesare al fiecărei unităţi de timp reale (aceasta este o aproximare, deoarece costul planificării şi al schimbărilor de context între procese trebuie, de asemenea, inclus în unitatea de timp reală). Mai precis, presupunem că unitatea centrală de prelucrare este inactivă şi *n* procese arbitrare $\{p_i \mid 0 \le i < n\}$ sunt gata de execuţie. De asemenea, (pentru uşurinţa notării) procesele apar în listă în ordinea indecşilor lor adică p_i apare înaintea lui p_j dacă i<j. Procesorul va fi alocat lui p_0, apoi lui p_1, apoi lui p_2,....,apoi lui p_{k-1}, apoi lui p_0 ş.a.m.d.

Există şi câteva opţiuni în implementarea algoritmului round-robin. Dacă procesorul termină execuţia unui proces înainte de expirarea cuantei de timp, imediat este apelat planificatorul de procese, care aloca procesorul unui alt proces. Când este creat un nou proces, acesta este plasat în lista proceselor gata de execuţie. Oricum, poziţia sa exactă în listă depinde de o opţiune de implementare: dacă lista este implementată ca o listă circulară, procesul nou creat este plasat în lista circulară imediat în spatele ultimului proces executat, aşa încât, celelalte *n-1* procese vor fi servite înaintea noului proces. O altă opţiune posibilă este implementarea listei sub forma unei cozi, în care procesele intră în ordine creării lor. Procesul nou creat intră în coadă la sfârşit, independent de

procesul care este în execuție la momentul apariției noului proces. Noul proces va aștepta, în medie, *n/2* cuante de timp, până când i se va aloca procesorul.

3. Modelarea performanțelor algoritmilor de planificare

Pentru modelarea performanțelor algoritmilor de planificare, vom utiliza următoarele notații:

- P=$\{p_i \mid 0 \le i < n\}$ un set de procese și $S(p_i)$ este starea procesului p_i, unde $S(p_i) \in$ {execuție, gata de execuție, blocat};
- Timpul de execuție, $T(p_i)$: timpul necesar execuției unui proces pentru a fi terminat;
- Timpul de așteptare, $W(p_i)$: timpul petrecut de proces în starea de gata de execuție înainte de prima lansare în execuție;
- Timpul de revenire a unui proces p_i, $T_r(p_i)$: timpul dintre momentul când procesul intră prima oară în starea de gata de execuție și momentul în care procesul iese din execuție pentru ultima oară.

4. Exemplificare

Să considerăm un exemplu practic de planificare prin algoritmii prezentați, urmând a examina performanțele obținute. Presupunem că avem 6 procese în lista de procese gata de execuție, ca în tabelul 1, și că ele au intrat în listă cu ordinea $p_0, p_1, p_2, p_3, p_4, p_5$. În exemplul considerat, presupunem că, în timpul servirii tuturor programelor din listă, nu mai sosesc și alte procese. Algoritmul *FIFO* va programa procesele ca în figura 1.

Tabelul 1. Timpii de execuție pentru 6 procese

Procesul	$T(P_i)$
0	325
1	150
2	450
3	245
4	90
5	120

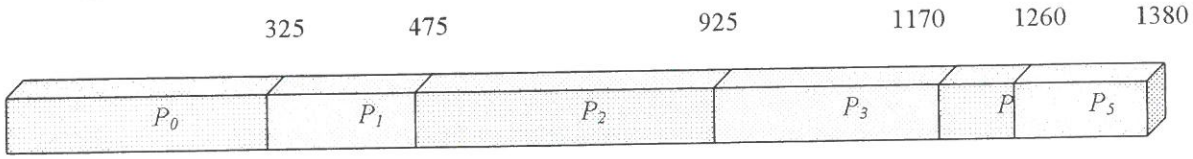


Figura 1. Planificarea FIFO

Putem determina timpul de reîntoarcere al fiecărui proces, observând planificarea *FIFO* din figura 1.

$$\begin{cases} T_r(p_0) = T(p_0) = 325 \\ T_r(p_1) = (T(p_1) + Tr(p_0)) = 150 + 325 = 475 \\ T_r(p_2) = (T(p_2) + T_r(p_1)) = 450 + 475 = 925 \\ T_r(p_3) = (T(p_3) + T_r(p_2)) = 245 + 925 = 1170 \\ T_r(p_4) = (T(p_4) + T_r(p_3)) = 90 + 1170 = 1260 \\ T_r(p_5) = (T(p_5) + T_r(p_4)) = 120 + 1260 = 1380 \end{cases}$$

Deci timpul mediu de reîntoarcere este:

Trm = (325+475+925+1170+1260+1380)/6 = 922,5

Se poate determina imediat şi timpul de aşteptare ca fiind:

$$\begin{cases} W(p_0)=0 \\ W(p_1)=T_r(p_0)=325 \\ W(p_2)=T_r(p_1)=475 \\ W(p_3)=T_r(p_2)=925 \\ W(p_4)=T_r(p_3)=1170 \\ W(p_5)=T_r(p_4)=1260 \end{cases}$$

Deci, timpul mediu de aşteptare este:

W = (0+325+475+925+1170+1260)/6 = 4155/5 = 831.

În continuare, vom studia cazul algoritmul *SJN*. Acesta va determina o planificare ca în figura 2. Din moment ce $T(p_4)=90$ este cel mai mic timp de servire, p_4 este planificat primul, iar pentru ca $T(p_5)=120$, p_5 este următorul şi aşa mai departe.

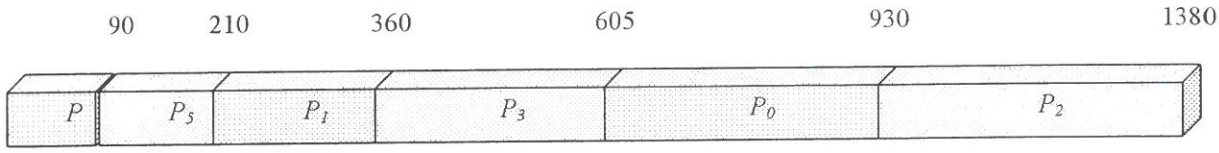


Figura 2. Planificarea SJN

Timpii de revenire sunt daţi de relaţiile:

$$\begin{cases} T_r(p_0)=T(p_0)+T(p_3)+T(p_1)+T(p_4)+T(p_5)=930 \\ T_r(p_1)=T(p_1)+T(p_4)+T(p_5)=360 \\ T_r(p_2)=T(p_2)+T(p_0)+T(p_3)+T(p_1)+T(p_4)+T(p_5)=1380 \\ T_r(p_3)=T(p_3)+T(p_1)+T(p_4)+T(p_5)=605 \\ T_r(p_4)=T(p_4)=90 \\ T_r(p_5)=T(p_4)+T(p_5)=210 \end{cases}$$

Astfel, timpul mediu de reîntoarcere este:

Trm = (930+360+1380+605+90+210)/6 = 596

Se observă că acesta este mai mic decât în situaţia precedentă, cea a planificării FIFO.

Determinăm în continuare timpii de aşteptare astfel:

$$\begin{cases} W(p_0)=605 \\ W(p_1)=210 \\ W(p_2)=930 \\ W(p_3)=360 \\ W(p_4)=0 \\ W(p_5)=90 \end{cases}$$

Deci, timpul mediu de aşteptare este:

W = (605+210+930+360+0+90)/6 = 366.

Pentru a analiza cazul planificării în funcţie de priorităţi, vom considera priorităţile ataşate fiecărui proces, ca şi în tabelul 2.

Tabelul 2 Timpii de execuţie pentru 6 procese		
Procesul	**$T(P_i)$**	**Prioritate**
0	325	1
1	150	6
2	450	3
3	245	4
4	90	2
5	120	5

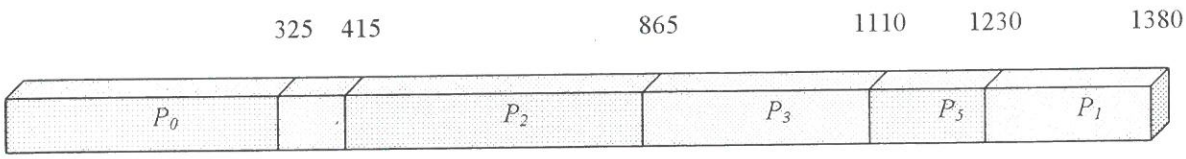


Figura 3. Planificarea pe bază de priorităţi

O cheie a performanţei unei planificări pe bază de priorităţi este alegerea priorităţii proceselor. Din nou, planificarea cu priorităţi poate cauza „înfometarea" proceselor cu prioritate joasă. Această „înfometare" poate fi compensată de calcularea interna a priorităţilor. Să presupunem că un parametru din funcţia de atribuire a priorităţii unui proces este durata de timp în care procesul a aşteptat accesul la unitatea centrală de prelucrare. Cu cât un proces aşteaptă mai mult, cu atât prioritatea sa devine mai mare. Această strategie tinde spre a elimina problema „înfometării". Timpul de reîntoarcere pentru exemplul din figura 3. va fi:

$$\begin{cases} T_r(p_0)=T(p_0)=325 \\ T_r(p_1)=T(p_0)+T(p_4)+T(p_2)+T(p_3)+T(p_5)+T(p_1)=1380 \\ T_r(p_2)=T(p_0)+T(T_4)+T(p_2)=865 \\ T_r(p_3)=T(p_0)+T(p_4)+T(p_2)+T(p_3)=1110 \\ T_r(p_4)=T(p_0)+T(p_4)=415 \\ T_r(p_5)=T(p_0)+T(p_4)+T(p_2)+T(p_3)+T(p_5)=1230 \end{cases}$$

Astfel, timpul mediu de reîntoarcere este:

T_{rm} = (325+1380+865+1110+415+1230)/6 = 5325/6 =.887,5

Determinăm timpii de aşteptare astfel:

$$\begin{cases} W(p_0)=0 \\ W(p_1)=1230 \\ W(p_2)=415 \\ W(p_3)=865 \\ W(p_4)=325 \\ W(p_5)=1110 \end{cases}$$

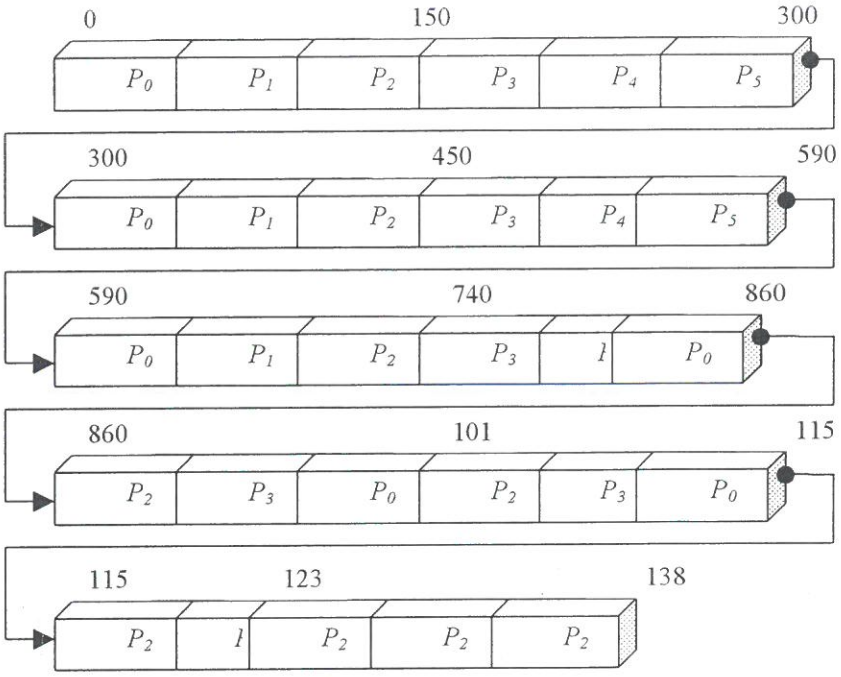
Deci, timpul mediu de aşteptare este:

Wm = (0+1230+415+865+325+1110)/6 = 3945/6 = 657,2

În continuare, vom analiza un exemplu de planificare round. În plus, vom considera cuanta de timp alocată pentru un proces la un moment dat ca fiind 50 de milisecunde. Într-o primă fază, timpul necesar unei schimbări de context între procese se consideră neglijabil.

Graficul care descrie rezultatul planificării proceselor este ilustrat în tabelul 3.

Tabelul 3. Modelul round robin şi indicatorii aferenţi

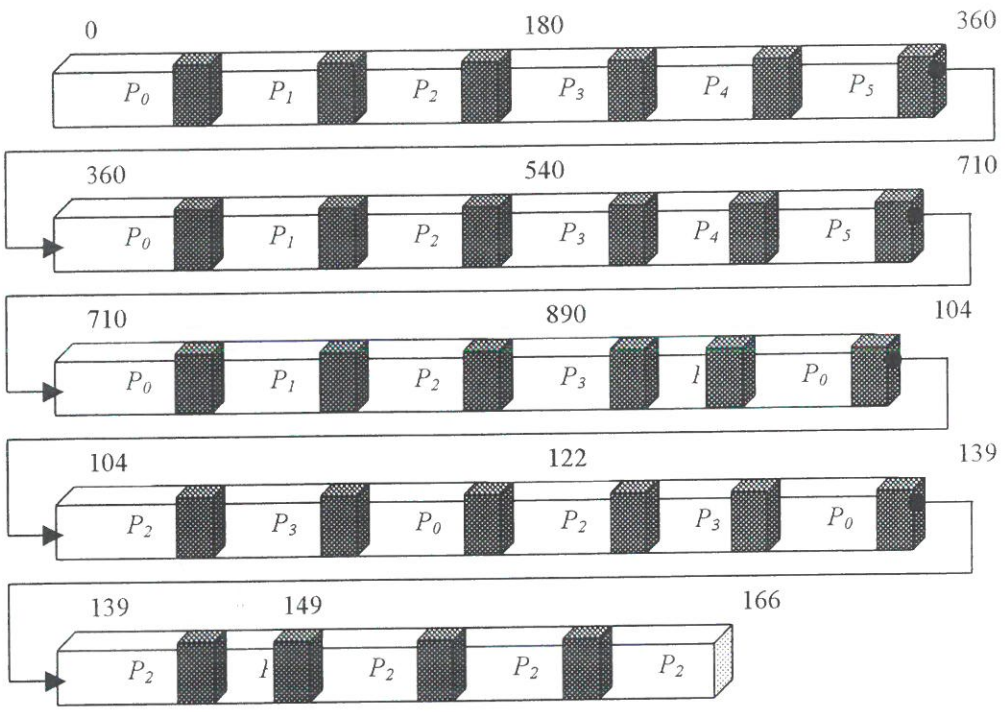
Indicatori	Planificatorul round robin
Timpul de revenire $T_r(p_0)$=1230 $T_r(p_1)$=690 $T_r(p_2)$=1380 $T_r(p_3)$=1105 $T_r(p_4)$=540 $T_r(p_5)$=810 T_rm =959	
Timpul de aşteptare W(p_0)=0 W(p_1)=50 W(p_2)=100 W(p_3)=150 W(p_4)=200 W(p_5)=250 Wm=125	

Timpul de aşteptare ilustrează beneficiul evident al algoritmul round-robin, în ceea ce priveşte rapiditatea cu care un proces este servit. Totuşi, timpul mediu de revenire nu diferă semnificativ de acela corespunzător unui algoritm nonpreemptiv.

Acum, reconsiderăm exemplul precedent, în care includem şi timpul de schimbare de context între două procese, şi presupunem că fiecare schimbare de context necesită 10 unităţi de timp (aşa cum reiese din tabelul 4.).

Tabelul 4. Modelul round robin cu timp de schimbare de context şi indicatorii aferenţi

Planificatorul round robin



Timpul de revenire	Timpul de aşteptare
$T_r(p_0)$=1490	W(p_0)=0
$T_r(p_1)$=820	W(p_1)=60
$T_r(p_2)$=1660	W(p_2)=120
$T_r(p_3)$=1295	W(p_3)=180
$T_r(p_4)$=640	W(p_4)=240
$T_r(p_5)$=970	W(p_5)=300
$T_r m$ =1146	Wm=150

5. Concluzii

Pentru toate aceste metode există abordări matematice coerente bazate pe calculul probabilistic. Pentru a estima cât mai bine funcţionalitatea unui algoritm real este recomandabil să încercăm abordarea probabilistică. Aceasta coroborată cu o eventuală simulare numerică pot scoate în evidenţă punctele tari şi cele slabe ale metodei analizate. Intrările şi ieşirile din sistem sunt testate în algoritm pe baza unor repartiţii corespunzătoare. Aceste repartiţii se apropie cel mai bine de cazul real. Abordarea matematică permite alegerea variantei de planificare optime.

Bibliografie

1. **DODESCU, G. D., S. NASTASE, F. C. PAPASTERE**: Sisteme de calcul şi operare, Editura Aldo, Bucureşti, 1995.
2. **DODESCU, G., A. VASILESCU**: Sisteme de operare Unix şi Windows, Editura Economică, Bucureşti, 2002.
3. **DODESCU, G., F. NASTASE**: Sisteme de calcul şi operare, Editura Economică, Bucureşti, 2002.
4. **TANENBAUM, A.**: Operanting Systems: Design and Implementation, Englewood Cliffs, NJ, Prentice Hall, 1997.