Sisteme de Operare

- Planificarea proceselor
  - Funcţionarea unui planificator
  - Implementarea unui planificator
  - Algoritmi de planificare

Planificarea proceselor

- Planificarea este funcţia principală a sistemelor de operare. Aproape toate resursele unui sistem de calcul sunt planificate înaintea utilizării
- Datorită naturii unor constrângeri relative la consumul de resurse, problema planificării se reduce la găsirea unui algoritm eficient pentru gestionarea accesului şi utilizarea resurselor pe baza unei metrici de performanţă.
- Tipurile de resurse luate în calcul sunt timpul CPU şi capacitatea memoriei.
- Clasificare:
  - planificare uniprocesor (task-uri cu resurse independente sau partajate)
  - planificare multiprocesor:
    - planificare statică
    - planificare dinamică

Planificarea proceselor

- problema planificării uniprocesor poate fi privită ca o problemă de căutare:
  - avem **n** procese **< p_1, p_2, ... p_n >** şi vrem să găsim o secvenţă de execuţie astfel încât toate procesele să se poată executa.
- Alocarea proceselor este diferită de planificare:
  - alocarea se referă la task-uri cu resurse independente, deci nu se realizează precedenţă între ele
  - planificarea se referă la procese care au resurse partajate deci şi relaţii de precedenţă.

Planificarea proceselor

- Planificarea statică:
  - există un set de procese < p_1, p_2, … p_n >.
  - Se obţine o planificare a proceselor pe sistemul multiprocesor dat în aşa numita fază de testare a fezabilităţii planificării, procesele se vor executa exact în ordinea stabilită de această planificare fără ca în timpul rulării proceselor să apară elemente necunoscute despre procese.
- Planificarea dinamică:
  - se pot varia caracteristicile (constrângerile) proceselor odată cu apariţia unor noi procese.

Deciziile de planificare

1. Când un proces trece din starea running în starea waiting (cerere I/O, crearea unui proces fiu sau aşteptarea terminării acestuia);
2. Când un proces trece din starea running în starea ready (când apare o întrerupere);
3. Când un proces trece din starea waiting în starea ready (terminarea unei operaţii I/O);
4. Când un proces este terminat.

- Doar în cazurile 1 şi 4, atunci planificarea este **nepreemptivă**;
- altfel, planificarea este **preemptivă**.

Planificarea nepreemptivă

- odată ce procesorul a fost alocat unui proces, procesul păstrează CPU până când se termină sau până când trece într-o stare de wait
- este folosită în cazul MSDOS, Windows 3.11
- nu necesită existenţa unui timer

Planificarea preemptivă

- necesită costuri suplimentare:
  - fie cazul a două procese A şi B care partajează o dată;
    - procesul A poate fi în mijlocul operaţiei de modificare a datei când este preemptat şi procesul B rulează;
    - procesul B poate încerca să citească data care în momentul respectiv nu este consistentă.
    - În acest caz este necesară introducerea de mecanisme suplimentare pentru coordonarea accesului la resursa comună.

Planificarea preemptivă şi kernelul

- are influenţă şi asupra proiectării kernelului sistemului de operare.
  - În timpul procesării unui apel sistem, kernelul poate fi ocupat cu o activitate asupra comportamentului unui proces, ceea ce poate duce la modificarea unor date ale kernelului (cozile I/O).
  - Ce se poate întâmpla dacă procesul este preemptat în mijlocul acestor modificări şi kernelul are nevoie să citească sau să modifice datele?

Planificarea preemptivă şi kernelul

- întreruperile pot interveni oricând
- nu pot fi tot timpul ignorate de kernel
- secţiunile de cod afectate de întreruperi trebuie protejate împotriva utilizării simultane (concurente) de mai multe procese.
  - se inhibă (disable) tratarea întreruperilor la intrarea în secţiunea critică (utilizarea datelor) şi se reactivează (enable) la ieşire .

Niveluri de planificare

- Planificarea pe termen lung
- Planificarea pe termen mediu
- Planificarea pe termen scurt

Planificarea pe termen lung

- are drept sarcină alegerea job-ului ce va fi executat, alocarea resurselor necesare şi crearea proceselor.
- rulează cu frecvenţa cea mai mică şi trebuie să separe tipurile de job-uri în funcţie de solicitări.
- La unele sisteme poate avea un rol minim sau poate lipsi (la sistemele time-sharing)

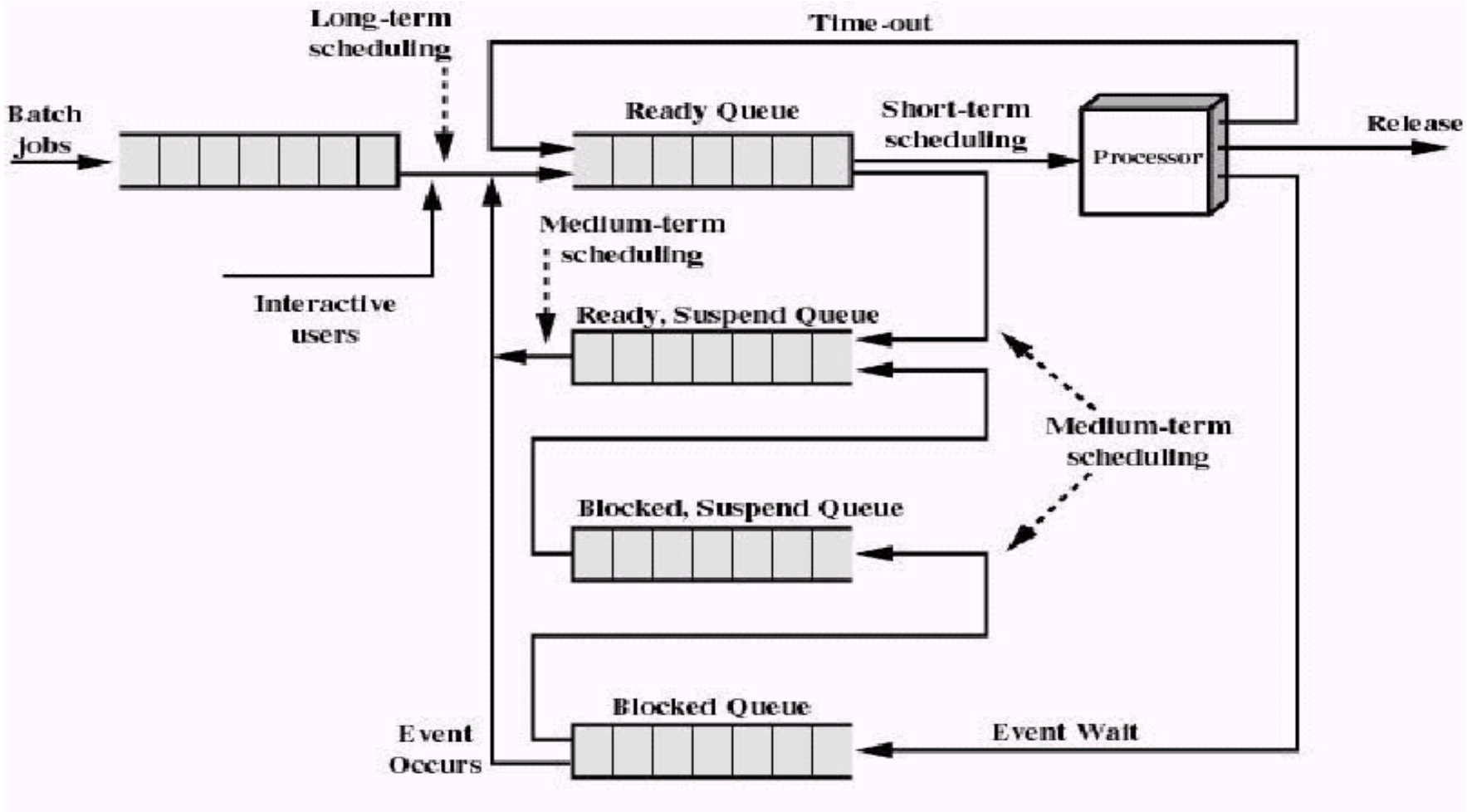
Planificarea pe termen mediu

- Planificatorul pe tremen mediu este cel care decide momentele în care se fac evaluări, procesele care se evacuează şi care se readuc în memorie pentru continuarea execuţiei.
- Stabilitatea sistemelor de tip time-sharing depinde de resursele sistemului de calcul şi de aceea la aceste sisteme se aplică tehnica de swapping prin care procesele sunt trecute din starea run în swap şi din swap în ready.
- Prin aceste evacuări temporare, gradul de multiprogramare scade şi se prelungeşte durata execuţiei, dar se permite accesul simultan al mai multor utilizatori în sistem.

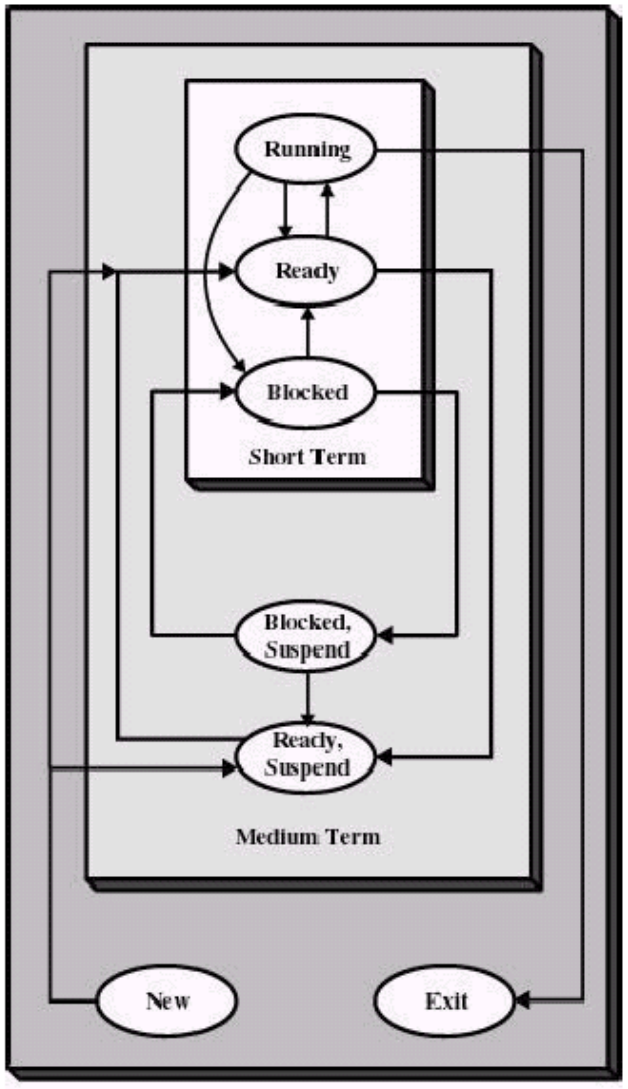
Planificarea pe termen scurt

- are în vedere trecerea alternativă a proceselor din starea **ready** în starea **run** şi invers.
- Tot la acest nivel se desfăşoară şi activitatea dispecerului de procesoare.

Cozile de aşteptare ale planificatorului



Stările proceselor şi tipul planificării



Planificare proceselor - Metrici

- **timpul de aşteptare** al unui proces: este timpul cât un proces aşteaptă în coada de execuţie

$$T_{wait} = \frac{\sum_{i=1}^{n} t_{wait}(p_i)}{n}$$

Planificare proceselor - Metrici

- **timpul de ciclare** reprezintă timpul din momentul creării procesului (intrare în coada gata de execuţie) până în momentul terminării execuţiei

$$t_{ciclare}(p_i) = t_{wait}(p_i) + \tau(p_i)$$

$\tau(p_i)$ - timpul de execuție

$$T_{ciclare} = \frac{\sum_{i=1}^{n} t_{ciclare}(p_i)}{n}$$ - timpul de ciclare mediu

încărcarea CPU

$$\rho = \alpha \cdot \frac{1}{\beta}$$

- α- rata medie de sosire a noilor procese în coada de execuţie
- β - rata medie de deservire a proceselor ($1/\beta$ este timpul mediu de execuţie)
- Dacă $\rho<1$ ($\alpha<\beta$) avem cazul în care se poate aplica un algoritm de planificare (încărcare normale) – stare stabilă.
- Dacă $\rho>1$ ($\alpha>\beta$), atunci unitatea centrală va fi saturată indiferent de algoritmul de planificare.
- Dacă $\rho=1$ ($\alpha=\beta$), lista de execuţie nu este suficient de lungă.
- Algoritmii ce vor fi prezentaţi se vor referi la cazul $\rho<1$ ($\alpha<\beta$).

Funcționarea unui planificator

- descrierea funcţionării unui planificator trebuie avute în vedere două aspecte:
  - regulile de acţiune
  - implementarea în contextul sistemului de operare.
- Fixarea sarcinilor unui planificator, indiferent de nivelul la care acţionează, se face precizând:
  - modalitatea de intervenţie
  - funcţia de prioritate
  - regula de arbitraj

Funcționarea unui planificator
Modalitatea de intervenție

- stabileşte momentele în care planificatorul intră în acţiune
  - momentele impuse de proces
    - atunci când un proces îşi termină activitatea sau când aşteaptă terminarea unor operaţii de I/O - planificatorul este partajat;
    - planificatorul este apelat de procese ca un subprogram
  - momentele impuse de SO
    - SO intervine indiferent de starea proceselor pe care le planifică – planificatorul este master.

Funcționarea unui planificator
Funcția de prioritate

- are ca argumente procesele şi parametrii sistemului.
- Determinarea priorităţii se face având în vedere criterii cum ar fi:
  - cererea de memorie,
  - atingerea unui timp de servire de către CPU,
  - timpul real din sistem,
  - timpul total de servire,
  - valorile priorităţilor externe,
  - necesarul de timp rămas până la terminarea procesului etc.

Funcționarea unui planificator
Funcția de prioritate

- Algoritmii de planificare trebuie să îndeplinească următoarele criterii:
  - să realizeze scopurile de performanţă pentru care au fost elaboraţi;
  - să aibă o durată foarte mică de execuţie, pentru a nu creşte în mod nejustificat timpul de execuţie alocat SO.
- Există multe metode matematice de planificare care dau soluţia optimă în nişte restricţii date
- Proiectanţii de sisteme de operare preferă algoritmi euristici, mai simpli şi cu rezultate mai mult sau mai puţin apropiate de cea optimă, deoarece un model sofisticat consumă mai mult timp făcând să crească timpul alocat SO, deci randamentul global să scadă.

Funcționarea unui planificator
Regula de arbitraj

- stabileşte o ordine în caz de priorităţi egale:
  - servirea în ordine cronologică
  - servirea circulară sau aleatoare.
- Observaţie: Informaţiile legate de starea proceselor care trebuiesc planificate sunt obţinute din PCB-ul fiecărui proces.

Implementarea unui planificator

- Toate tipurile de planificatoare sunt implementate prin intermediul semafoarelor şi folosesc facilităţile oferite de gestiunea memoriei.
- În funcţie de tipul sistemului de operare, planificatoarele deţin module specializate de alocare şi eliberare a resurselor, prevenire, detectare şi ieşire din blocaje.
- Conceptele de multiprogramare şi programare în timp real sunt implementate cu ajutorul planificării pe termen scurt.
- Tot la acest nivel are loc selectarea proceselor candidate la resursele disponibile.
- La nivel mediu este potrivit să se modifice priorităţile proceselor, dacă SO permite această facilitate.

Algoritmi de planificare

Criterii de evaluare

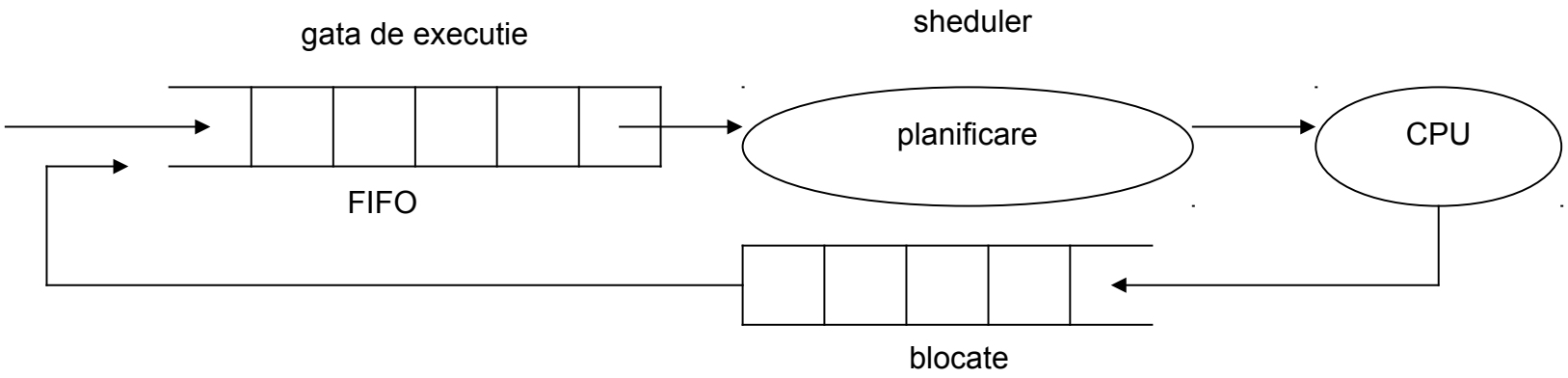
- Utilizarea CPU
- Răspunsul (troughput)
- Numărul de procese terminate în unitatea de timp
- Timpul de ciclare (turnaround) – timpul mediu de la sosirea unui proces până la terminarea lui
- Timp de aşteptare (waiting time) în lista gata de execuţie
- Timpul de răspuns (response time) – timpul mediu de la sosirea proceselor până la prima execuţie
- Eficienţa planificatorului = overhead-ul introdus de planificator.
- Reprezentarea proceselor se va face cu ajutorul diagramelor Gantt.

Algoritmi de planificare
FCFS (First-Come, First-served)

- Procesele sunt planificate pe măsura sosirii lor în coada gata de execuţie.
- Procesorul este alocat procesului care îl cere primul.
- Algoritmul mai este denumit şi FIFO.
- Unitatea de măsură a performanţei este timpul mediu de aşteptare.
- În acest caz **pri(i) = t_i** , unde t_i este momentul când soseşte în coada gata de execuţie. Dacă pentru două procese **i** şi **j**, avem **pri(i) < pri(j)** atunci **t_i < t_j** .
- În general vom spune ca un proces este de prioritate cu atât mai mare cu cât valoarea **pri(i)** este mai mică.

Algoritmi de planificare
FCFS (First-Come, First-served)

- Pentru FCFS planificarea se va face în funcţie de prioritatea maximă. Lista gata de execuţie va fi o listă FIFO.



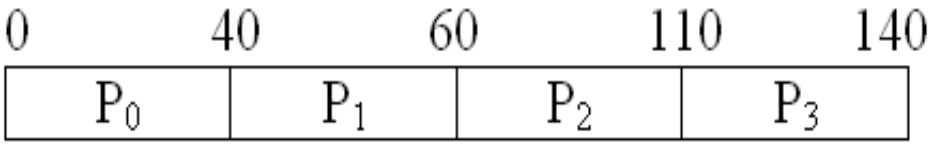
Algoritmi de planificare

FCFS (First-Come, First-served)

Exemplu:

procese	timp de execuție τ	prioritate
0	40	4
1	20	2
2	50	1
3	30	3

Planificare:



$t_{wait}(0) = 0, \quad t_{ciclare}(0) = 40$

$t_{wait}(1) = 40, \quad t_{ciclare}(1) = 60$

$t_{wait}(2) = 60, \quad t_{ciclare}(2) = 110$

$t_{wait}(3) = 110, \quad t_{ciclare}(3) = 140$

$T_{wait} = (0 + 40 + 60 + 110)/4 = 52,5$

$T_{ciclare} = (40 + 60 + 110 + 140)/4 = 87,5$

$t_{out} = 140/4 = 35 \Rightarrow$ în medie 35 de unități de timp pentru fiecare proces care trebuie să se execute.

Algoritmi de planificare

Shortest-Job-First (SJF) nepreemptiv

- Procesul planificat pentru execuţie este procesul cu timpul de execuţie cel mai mic. Acest algoritm se mai numeşte şi **Shortest-Job-Next (SJN)**.

Pentru cazul anterior avem:

$t_{wait}(1) = 0, \quad t_{ciclare}(1) = 20$

$t_{wait}(3) = 20, \quad t_{ciclare}(3) = 50$

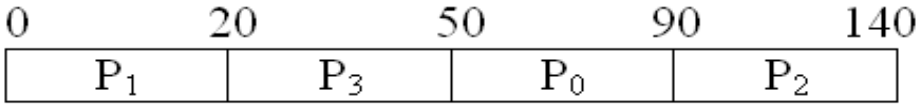
$t_{wait}(0) = 50, \quad t_{ciclare}(0) = 90$

$t_{wait}(2) = 90, \quad t_{ciclare}(2) = 140$

$T_{wait} = (0 + 20 + 50 + 90)/4 = 40$

$T_{ciclare} = (20 + 50 + 90 + 140)/4 = 75$

Planificare:

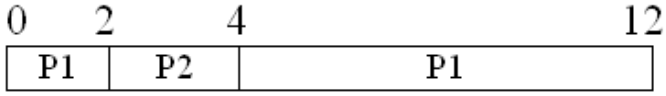


Observaţie: Timpul total este acelaşi dar timpul mediu de aşteptare este mai mic. Acest algoritm este optim dacă toate job-urile ajung în acelaşi timp.

Algoritmi de planificare
Shortest-Job-First (SJF) preemptiv

- Procesul planificat pentru execuţie este procesul cu timpul de execuţie rămas cel mai mic. Acest algoritm mai este numit şi **Shortest Remaining Time First**.

procese	Timp de execuţie τ	prioritate	Timp sosire
P1	10	1	0
P2	2	2	2

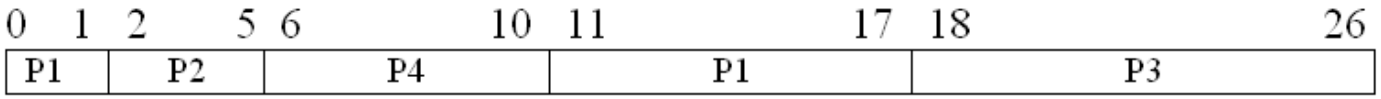


$t_{wait}(1) = 4\text{-}2=2$

$t_{wait}(2) = 0$

$T_{wait} = (0 + 2)/2 = 1$

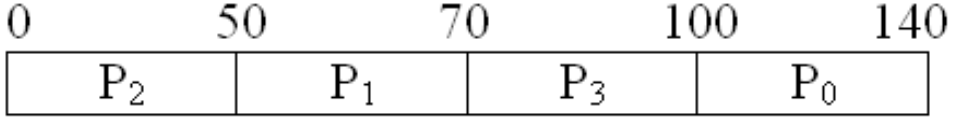
procese	Timp de execuţie τ	Timp sosire
P1	8	0
P2	4	1
P3	9	2
P4	5	3



Algoritmi de planificare
PS (Priority Scheduling)

- Fiecare proces are asociată prioritate, fiind lansate în execuţie de la prioritatea cea mai mică la prioritatea cea mai mare.
- Este cel mai folosit algoritm.

Planificare:



$T_{wait} = (0 + 50 + 70 + 100) / 4 = 55$

$T_{ciclare} = (50 + 70 + 100 + 140) / 4 = 90$

În acest caz, algoritmul funcționează cu *priorități statice*.

Algoritmi de planificare
PS (Priority Scheduling)

- Se poate introduce un mecanism de priorităţi dinamice, în care pe măsură ce timpul de aşteptare creşte, va creşte şi prioritatea.
- Această situaţie este întâlnită la Unix - se opreşte periodic (la fiecare secundă) activitatea sistemului şi se recalculează prioritatea fiecărui proces.
- Astfel se garantează un timp mediu de răspuns rezonabil pentru fiecare proces din sistem, dar nu se asigură răspuns prompt la o execuţie secvenţială.
- Priorităţile se stabilesc astfel:
  - job-ul primeşte prioritatea la intrarea în sistem şi o păstrează până la sfârşit (este posibil să apară fenomenul de "înfometare", dacă apar multe procese cu prioritate mare);
  - SO calculează priorităţile după reguli proprii şi le ataşează dinamic proceselor în execuţie. Această variantă este folosită la planificarea pe termen mediu.

Algoritmi de planificare
Round-Robin (RR)

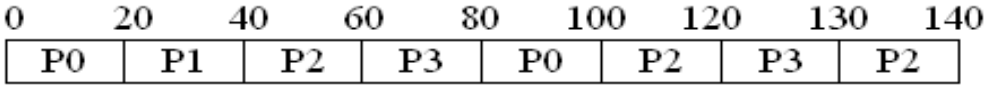
- este un algoritm preemptiv destinat sistemelor de tip **time-sharing** şi se bazează pe distribuirea în mod egal a timpului de procesare între procese.
- Este folosită o **cuantă de timp q** (cu valori între 10 şi 100 milisecunde) pe durata căreia sunt executate pe rând părţi din fiecare proces.
- Dacă se introduce şi **timpul consumat c** prin schimbarea contextului, fiecare proces va primi de fapt **c+q** unităţi de timp.
- Unele procese se pot termina înainte de expirarea cuantei de timp, moment în care se invoca planificatorul care reface priorităţile, resetează cuanta de timp şi replanifică procesele.
- Replanificarea are loc şi la apariţia unui proces nou.
- Algoritmul RR se implementează folosind întreruperea de ceas a sistemului respectiv.

Algoritmi de planificare
Round-Robin (RR)

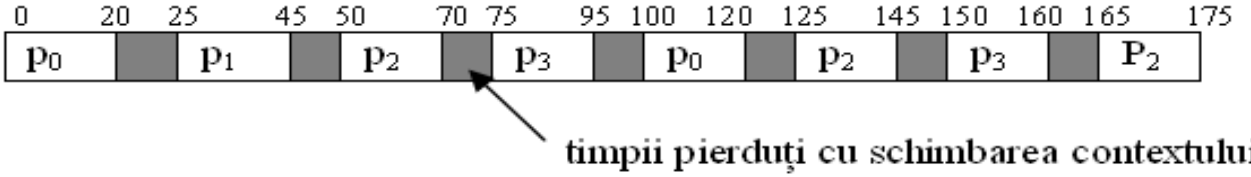
- Dacă cuanta de timp q este mai mare algoritmul tinde către FCFS.
- Algoritmul RR funcţionează cel mai bine când 80% din procese au timpii de execuţie mai mari decât cuanta **q**.

Procese	timp de execuţie τ
0	40
1	20
2	50
3	30

Fie $c = 0$ şi $q = 20$



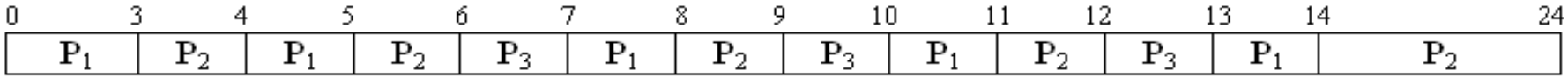
$t_{ciclare} = (100+40+140+130)/4=102.5;$
$t_{wait} = (60+20+40+40+10+60+40)/4=67.5;$
Dacă consideram c=5, q=20 avem:



Algoritmi de planificare
Round-Robin (RR)

procese	Timp de execuție τ	Timp sosire
P1	7	0
P2	14	3
P3	3	6

Fie $c = 0$ și $q = 1$



Algoritmi de planificare

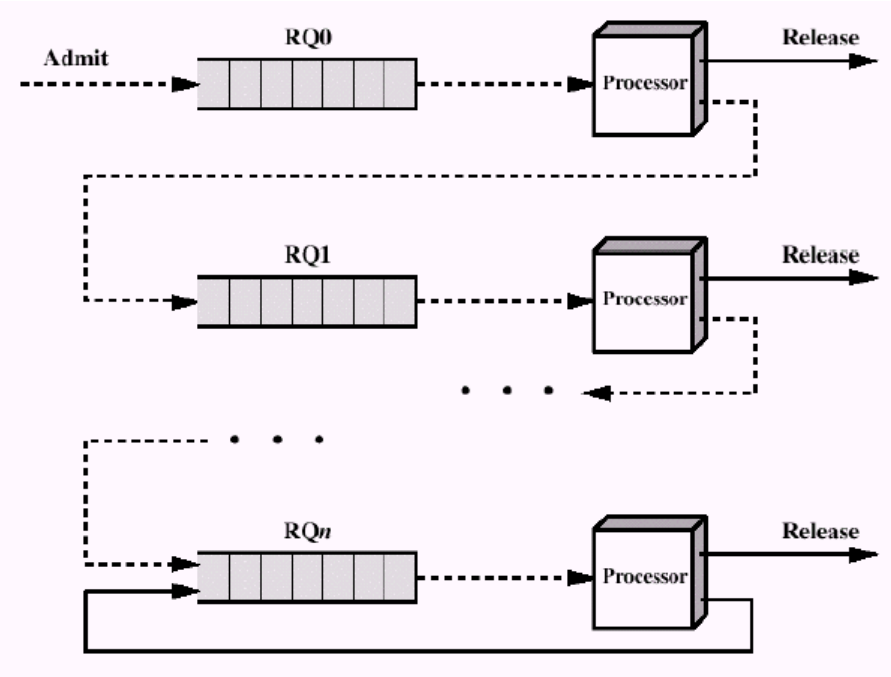
Highest Response Ratio Next (HRRN)

- Este ales procesul cu cea mai mare rată de deservire (response ratio – rr):
- În general sunt favorizate procesele cu durata de execuţie cea mai mică. Dacă aşteptarea procesului creşte, atunci creşte şi valoarea ratei de deservire.

$$rr = \frac{time\ spent\ waiting + expected\ service\ time}{expected\ service\ time}$$

Algoritmi de planificare

- **Algoritmul Feedback**

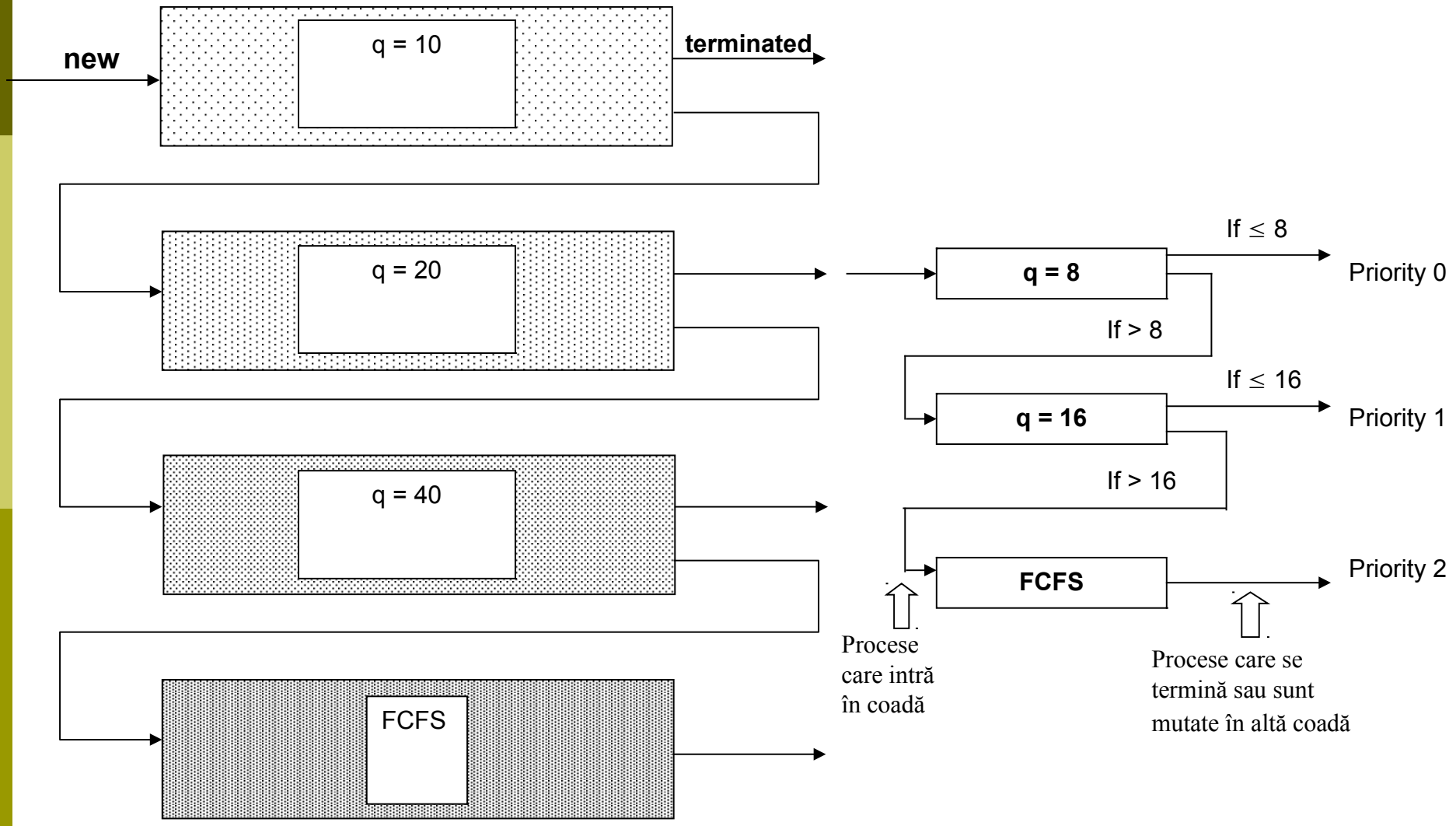


- este folosit atunci când nu se cunoaşte timpul de care mai are nevoie un proces ca să-şi termine execuţia.
- Sunt penalizate procesele care rulează prea mult şi poate duce la apariţia fenomenului de "înfometare" (process/ resource starvation) dacă nu variem algoritmii de planificare şi priorităţile în funcţie de cozile de aşteptare.

Algoritmi de planificare

- **Planificarea cu listele multinivel**
  - Este o combinaţie între:
    - algoritmii bazaţi pe priorităţi,
    - Round-Robin şi
    - algoritmi folosiţi pentru tratarea proceselor de aceeaşi prioritate (HRRN).
- Presupunem că lista gata de execuţie este formată din **n** subliste în care procesele au priorităţile între **1** şi **m**.
- În acest caz procesul $\mathbf{P_i}$ din sublista **k** va avea prioritatea **k**.
- Pentru a înlătura dezavantajul unui timp de aşteptare mare pentru subliste apropiate de **n** se poate folosi o schemă de planificare care să favorizeze subliste de mare prioritate (algoritmi nepreemptivi în liste şi între liste algoritmi preemptivi).

Planificarea cu listele multinivel



Inversarea priorității

- apare atunci când un proces de prioritate scăzută accesează o secţiune critică apoi un proces de prioritate mare accesează şi el SC respectivă şi se blochează.
- procese de prioritate intermediară vor împiedica de asemenea primul proces (de prioritatea cea mai scăzută) să deblocheze secţiunea critică.

Prevenirea inversării priorităţii

- Se foloseşte “moştenirea priorităţii”:
  - de fiecare dată când un proces deţine o SC pentru care aşteaptă şi alte procese i se acordă respectivului proces maximul priorităţii proceselor aflate în aşteptare.
  - Problema este că moştenirea priorităţii micşorează eficienţa algoritmului de planificare şi creşte overhead-ul (i se da prioritate maxima ca să nu fie preemptat şi să termine lucrul cu SC).

Prevenirea inversării priorității

- Preempţia poate interacţiona cu sincronizarea într-un context multiprocesor generând un alt efect nedorit numit **efectul de convoi**:
  - un proces accesează o secţiune critică după care se suspendă.
  - Alte procese care au nevoie de secţiunea critică vor trebui suspendate până când primul se va trezi (va fi reactivat) şi va termina secţiunea critică.
  - În acel moment toate procesele suspendate din cauza sincronizării vor fi trezite încercând pe baza priorităţii să acceseze secţiunea critică.
  - Astfel, se creează efectul de convoi.
- În general, **efectul de convoi** apare când o mulţime de procese au nevoie de o resursă pentru un timp scurt, iar un altul deţine resursa pentru un timp mult mai lung blocându-le pe primele.