Planificarea execuţiei proceselor

Introducere

Structura planificatorului

Strategiile fără evacuare

Strategii cu evacuare

Introducere

- Scopul planificării execuţiei proceselor este de a le aloca procesorul (CPU) sau procesoarele în vederea execuţiei lor, astfel încât să fie îndeplinite anumite obiective ale sistemului:

  - timp de răspuns cât mai mic,

  - cresterea productivităţii, exprimată prin numărul de procese sau thread-uri executate pe unitatea de timp (oră);

  - creşterea gradului de utilizare a CPU.
- Planificariea execuţiei proceselor poate fi împărţită în:

  - planificare pe termen lung;

  - planificare pe termen mediu;

  - planificare pe termen scurt.
- **Planificare pe termen lung**
- Se referă la decizia de a adăuga un nou proces pentru a fi executat, pe lângă cele care se află în execuţie (starea **new**).
- Astfel, se controlează gradul de multiprogramare. Odată admis, un program sau job devine proces si el este adaugat la coada proceselor planificate pe termen scurt.

- În această situaţie sistemul trebuie să ia două decizii:

  1. Când sistemul de operare poate să ia în considerare unul sau mai multe programe sau job-ri care să devină procese; o astfel de decizie ţine cont de gradul de multiprogramare. Limitarea gradului de multiprogramare de către planificator urmăreşte să asigure proceselor în curs de execuţie un timp rezonabil pe care să îl petreacă în sistem. De fiecare dată când se termină execuţia unui proces, planificatorul decide dacă este cazul să creeze încă unul sau mai multe procese, dacă procentul de timp de lenevire al CPU depăşeşte un anumit prag.

  2.Care dintre programe sau job-uri să devină procese. Această decizie se poate lua pe baza mai multor criterii:

  - împărţirea pe clase de priorităţi a programelor;

  - ce resurse cerute sunt disponibile la momentul respectiv.

- De **exemplu**, într-un sistem bazat pe partajarea timpului, atunci când un utilizator încearcă să se logineze, cererea poate să fie respinsă, deoarece cererea de execuţie a procesului de loginare nu poate să fie satisfăcută din cauza suprasaturării sistemului, urmând ca cererea respectivă să fie luată în consideraţie după o nouă încercare de loginare a utilizatorului.

- **Planificare pe termen mediu.**
- Se referă la decizia de a adăuga un nou proces pentru a fi încărcat în memorie în vederea execuţiei-starea **ready**.
- Se poate realiza din diverse motive:

  - trecerea din starea new în starea ready;

  - revenirea din starea de blocare;

  - reîncărcarea unui proces care a fost salvat pe disc(swaped); se întâlnea mai ales la sistemele care nu aveau memorie virtuală.

- **Planificarea pe termen scurt**.
- Se referă la decizia de a selecta unul dintre procesele aflate în starea **ready** pentru a primi serviciile CPU.
- Obiectivul principal al planificării pe termen scurt este de a aloca timp procesor, astfel încât să se optimize unul sau mai multe aspecte ale funcţionării sistemului de calcul.
- În acest sens, sunt stabilite unul sau mai multe criterii, pe baza cărora se evaluează politicile de planificare. Aceste criterii pot fi orientate către utilizator sau către sistem.
- **Criterii orientate către utilizator.**

  - **Timpul petrecut în sistem de proces-** intervalul de timp dintre momentul transmiterii procesului şi cel al terminării execuţiei; include timpul necesar execuţiei, plus timpul petrecut pentru a aştepta la diverse resurse, inclusive procesorul. Este potrivit mai ales pentru sistemele cu prelucrare în loturi.

  - **Timpul de raspuns**- este indicat ptr. sistemele interactive şi reprezintăn intervalul de timp dintre momentul lansării procesului şi cel al terminării lui.

  - **Termenul limită** la care un proces trebuie executat. Când este prevăzut, celelalte criterii sunt subordonate**.**

- **Criterii orientate către sistem.**

  - **Productivitatea**-reprezintă numărul de procese executate pe unitatea de timp.

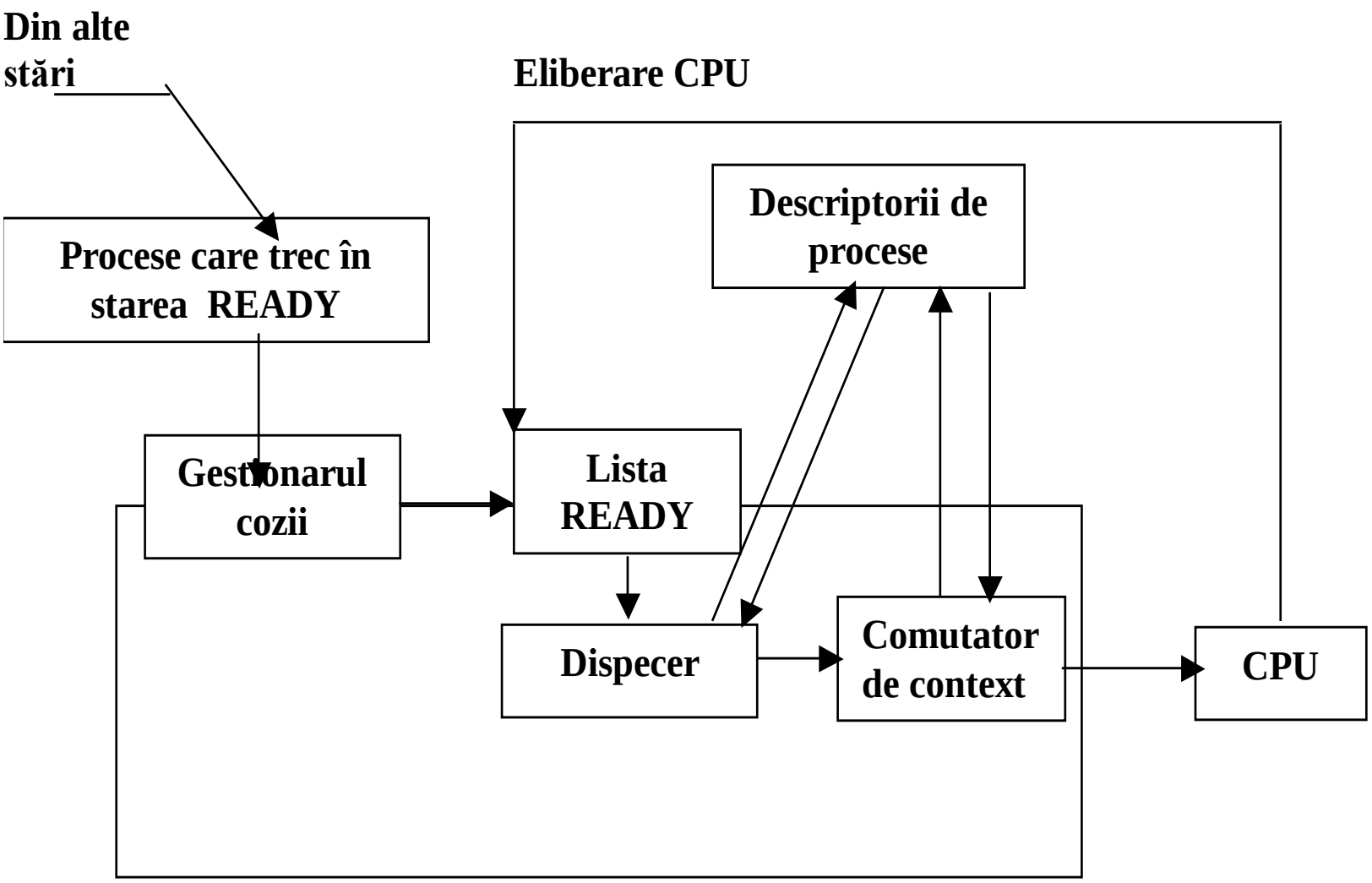
  - **Gradul de utilizare al procesorului** - procentul de timp în care procesorul este ocupat.

  - **Înfometare**- timpul de aşteptare al proceselor pentru a fi servite de processor este suficient de mic.

  - **Utilizarea priorităţilor** -procesele cu prioritate mai înaltă să fie servite preferenţial de CPU.

  - **Încărcarea (Utilizarea) echilibrată a resurselor** -resursele să nu fie ocupate ptr. un interval de timp mare (sunt implicate şi celelalte strategii de planificare).

Structura planificatorului



- **Gestionarul cozii proceselor** care aşteptă să primească serviciile CPU.
- Când un proces trece în starea READY, descriptorul său este prelucrat şi este introdus într-o coadă înlănţuită numită **lista READY**; această coadă este organizată pe priorităţi, care sunt calculate de către gestionarul cozii, ori la introducerea procesului în coadă sau în momentul selectării unui proces pentru a fi servit de către CPU.
- **Comutatorul de context (CS-C**ontext **S**witcher**)** salvează conţinuturile tuturor regiştrilor CPU(PC, IR, starea procesului, starea ALU etc.) relative la procesul în execuţie, în momentul când planificatorul comută CPU de la execuţia procesului respectiv. la un alt proces. Există două modalităţi prin care se execută această comutare de context:

  - voluntară, care presupune că procesul care deţine CPU, cere el însuşi eliberarea acesteia pentru a fi acordată unui alt proces.

  - involuntară, care presupune că eliberarea CPU se face prin intermediul unei întreruperi. Această metodă presupune existenţa unui **dispozitiv de măsurare a intervalelor de timp**, care să genereze o întrerupere oricând expiră intervalul de timp alocat procesului. O astfel de metodă de planificare se mai numeşte şi planificare cu evacuare sau cu forţare.

- **Dispecerul** este cerut pentru a selecta un nou proces aflat în lista READY, pentru a fi servit de către CPU, indiferent de modalitatea de eliberare a CPU. Acest lucru se poate realiza în două moduri:

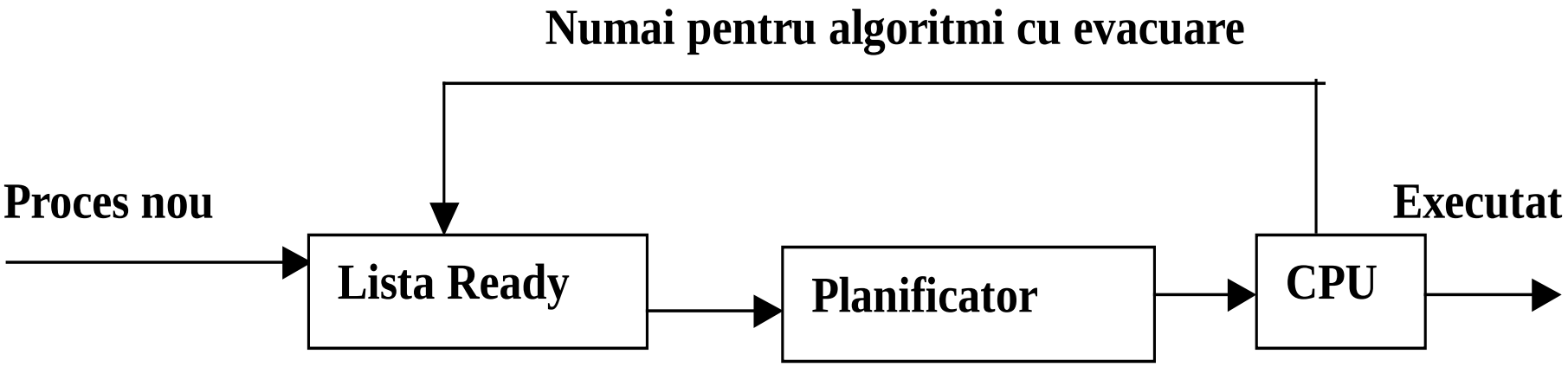
  • Procesul eliberează în mod voluntar CPU, când face o cere de resursă, trecând astfel controlul administratorului resurselor; CPU fiind eliberată, alte procese o pot utiliza. După terminarea utilizării resursei cerute, procesul revine în starea READY. Dacă resursa cerută nu este disponibilă, procesul este blocat (starea WAIT), până când resursa devine liberă, iar după alocarea şi utilizarea ei trece în starea READY.

  • Procesul poate fi evacuat în mod involuntar din utilizarea procesorului.
- **Salvarea contextului procesului.** Când CPU este multiplexată, apar două tipuri de comutare a contextului. În primul caz, se realizează salvarea contextului procesului care este în curs de execuţie de către SO şi este încărcat cel al dispecerului. A doua situaţie constă în cedarea CPU de către dispecer şi încărcarea procesului care urmează să fie servit de către CPU. Salvarea contextului este o operaţie destul de costisitoare, care poate afecta performanţele. Dacă notăm cu n, numărul regiştrilor generali şi cu m numărul regiştrilor de stare şi dacă presupunem că pentru a salva un singur registru sunt necesare b operaţii, fiecare instrucţiune de stocare necesitând K unităţi de timp, atunci pentru salvarea stării procesului sunt necesare $(n+m)bK$ unităţi de timp.

- **Performanţe.** Planificatorul are o influenţă majoră asupra performanţelor unui calculator, care lucrează în regim de multiprogramare, deoarece el decide când un proces este servit de către CPU. Dacă momentul când un proces este selectat să fie servit, tinde către momentul când este introdus în lista READY, atunci putem spune că el va fi servit aprope când doreşte. În schimb, dacă un proces este neglijat de către dispecer(fenomenul de „înfometare”), atunci va sta destul de mult în starea READY, deci timpul său de execuţie va fi destul de mare. De asemenea, performanţele planificatorului sunt influenţate de timpul necesar pentru comutarea contextului. Toate aceste aspecte sunt influenţate atât de componentele hardware, cât şi de cele software, reprezentate de componenta de planificare a SO.
- **Notaţii**. În cele ce urmează vom nota cu: $P = \{p_0, .., p_{n-1}\}$ mulţimea celor *n* procese din sistem; $S(p_i)$ este starea procesului $p_i, i = 0,..,n\text{-}1, S(p_i) \in \{RUN, READY, WAIT\}$ De asemenea, $\tau(p_i)$ este timpul efectiv de servire, adică intervalul de timp cât procesul p_i va fi în starea RUN; $W(p_i)$ este intervalul de timp petrecut de proces în starea READY pâna la prima tranziţie în starea RUN. De asemenea, $T(p_i)$ reprezintă timpul total petrecut în sistem de proces, din momentul când procesul intră pentru prima dată în starea READY, până când este executat.

Un model simplificat de planificare a proceselor

- Cererile de servicii adresate unităţii centrale se intercalează cu cereri de alocări de resurse. Modelul de planificare al proceselor poate fi simplificat, neluând în consideraţie efectele datorate competiţiei pentru alocarea resurselor, exceptând CPU(figura urm). În cadrul acestui model, procesul poate fi numai într-una din stările RUN sau READY.



Strategiile fără evacuare

- **Strategiile fără evacuare** permit unui proces ca atunci când îi este alocată CPU să fie executat complet, deci nu mai există procesul de comutare şi nici cel de introducere şi extragere din lista READY. Aceste metode se bazează pe modelele de aşteptare.
- **Algoritmul FCFS(First Come First Served)** sau **FIFO (First In First Output)**. Conform acestui algoritm, lucrările sunt servite în ordinea lor cronologică în care cer procesorul.
- Lista READY este organizată pe principiul FIFO, administratorul cozii adaugă lucrări la sfârşitul cozii, iar dispecerul scoate lucrările pentru a fi servite de CPU din capul cozii. Este un algoritm simplu, dar nu foarte eficient.
- **Exemplu.** Să presupunem că există *5* joburi în sistem, p_0, p_1, p_2, p_3, p_4 cu timpii de execuţie *350*, *125*, *475*, *250* respectiv *75*. Dacă acestea sosesc în ordinea prezentată, ordinea temporală a execuţiei lor, este reliefată prin diagrama Gantt din figura urm. Avem

$$T(p_0) = \tau(p_0) = 350 \quad T(p_1) = \tau(p_1) + T(p_0) = 125 + 350 = 475$$

$$T(p_2) = \tau(p_2) + T(p_1) = 475 + 475 = 950$$

$$T(p_4) = \tau(p_4) + T(p_3) = 75 + 1200 = 1275$$

$$T(p_3) = \tau(p_3) + T(p_2) = 250 + 950 = 1200$$

- Media timpilor petrecuţi în sistem va fi deci

$$T = (350 + 475 + 950 + 1200)/5 = 850$$

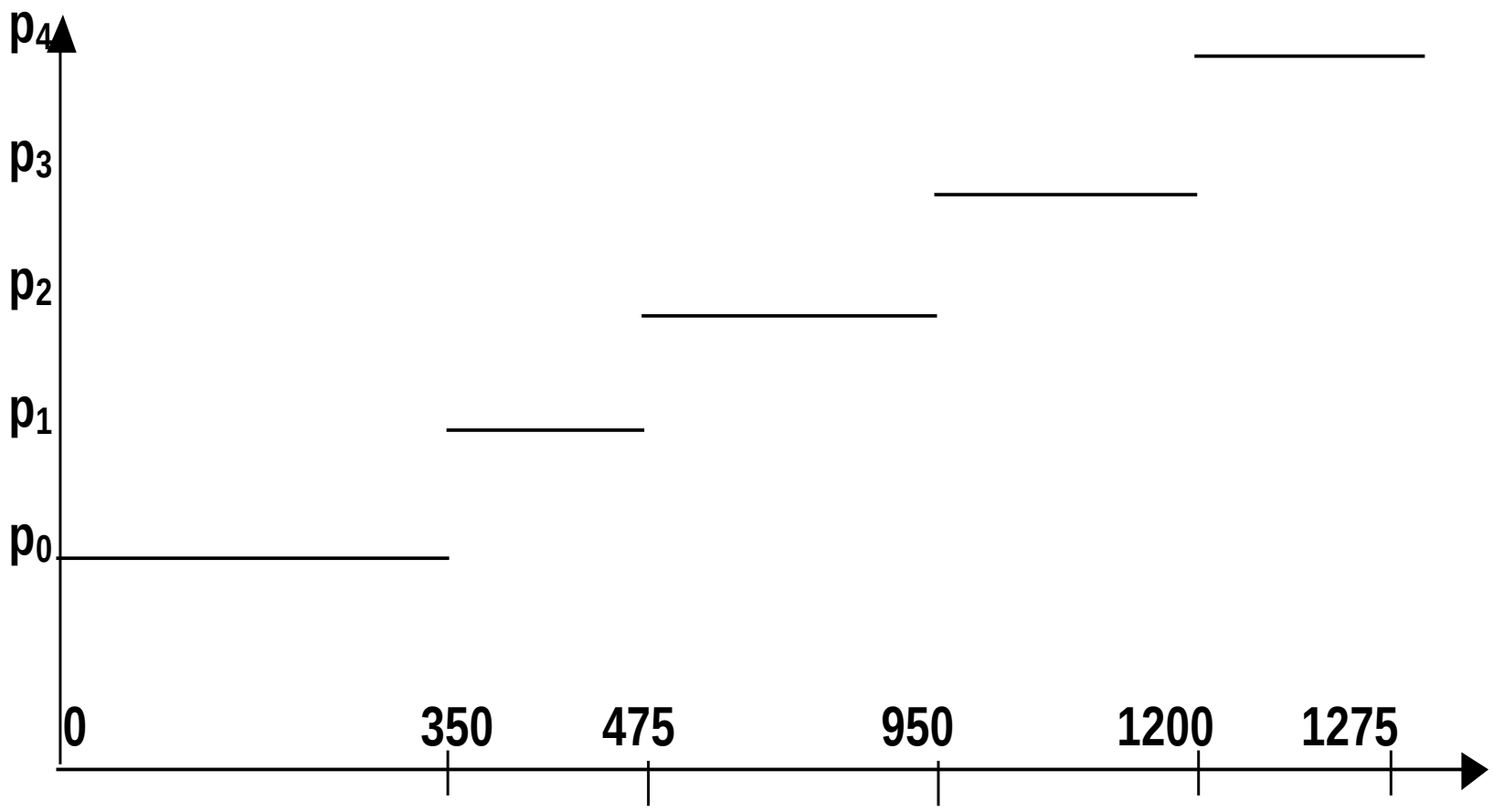
- De asemenea timpii de aşteptare vor fi: $W(p_0) = 0$

$$W(p_1) = T(p_0) = 350 \qquad W(p_2) = T(p_1) = 475$$

$$W(p_3) = T(p_2) = 950 \qquad W(p_4) = T(p_3) = 1200$$

- Deci media timpului de aşteptare până la servire, va fi

$$W = (0 + 350 + 475 + 950 + 1200)/5 = 595$$


p4
p3
p2
p1
p0
0
350
475
950
1200
1275

Algoritmul SJN (Shortest Job Next)

- În exemplul anterior, să considerăm că ordinea de sosire este p_4, p_1, p_2, p_3, p_0 ; vom avea: $T(p_4) = \tau(p_4) = 75$

$$T(p_1) = \tau(p_1) + T(p_4) = 125 + 75 = 200$$
$$T(p_2) = \tau(p_2) + T(p_1) = 200 + 475 = 675$$
$$T(p_3) = \tau(p_3) + T(p_2) = 250 + 675 = 925$$
$$T(p_0) = \tau(p_0) + T(p_3) = 75 + 925 = 1000$$

- Media timpilor petrecuţi în sistem va fi deci

$$T = (75 + 200 + 675 + 925 + 1000)/5 = 2875/5 = 575$$

- De asemenea timpii de aşteptare vor fi: $W(p_4) = 0$

$$W(p_1) = T(p_4) = 75 \quad W(p_2) = T(p_1) = 200$$
$$W(p_3) = T(p_2) = 675 \quad W(p_1) = T(p_3) = 925$$

- Deci media timpului de aşteptare până la servire, va fi

$$W = (0 + 75 + 200 + 675 + 925)/5 = 1875/5 = 355$$

- **Obs.** Mediile timpilor petrecuţi în sist. precum şi a timpilor de aşteptare sunt mai mici în cazul al doilea. Aceste medii descresc, dacă mărimea timpilor de servire respectă ordinea de sosire.

- **Algoritmul SJN** se bazează pe observaţia anterioară. De fiecare dată, jobul care se execută primul este cel care consumă cel mai puţin timp CPU. Probabil că acest algoritm este cel mai bun, însă are dezavantajul că ar trebui să se cunoască dinainte timpul CPU necesar al fiecărui proces.
- **Exemplu**. Considerând procesele prezentate în exemplul anterior, conform acestei metode ordinea de execuţie este p_4, p_1, p_3, p_0, p_2 , reliefată în diagrama Gantt din figura urm. şi atunci:

$$T(p_4) = \tau(p_4) = 75 \qquad T(p_1) = \tau(p_1) + T(p_4) = 125 + 75 = 200$$
$$T(p_3) = \tau(p_3) + T(p_1) = 250 + 200 = 450$$
$$T(p_0) = \tau(p_0) + T(p_3) = 350 + 450 = 800 \qquad T(p_2) = \tau(p_2) + T(p_0) = 475 + 800 = 1275$$

- Media timpilor petrecuţi în sistem va fi deci

$$T = (75 + 200 + 450 + 800 + 1275)/5 = 2450/5 = 490$$

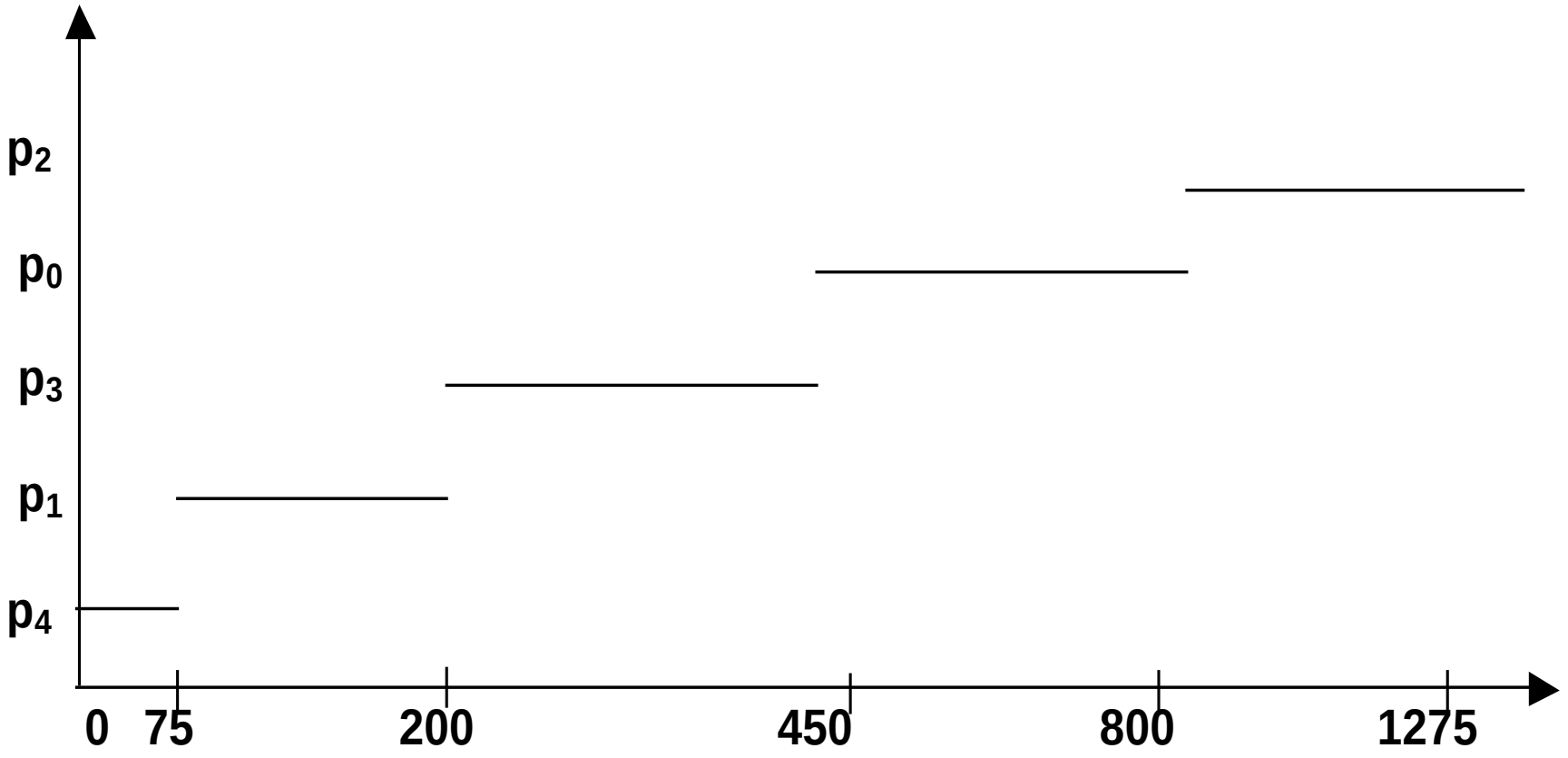
- De asemenea timpii de aşteptare vor fi: $W(p_4) = 0$

$$W(p_1) = T(p_4) = 75 \qquad W(p_3) = T(p_1) = 200 \qquad W(p_0) = T(p_3) = 450$$
$$W(p_2) = T(p_0) = 800$$

- Deci media timpului de aşteptare până la servire, va fi

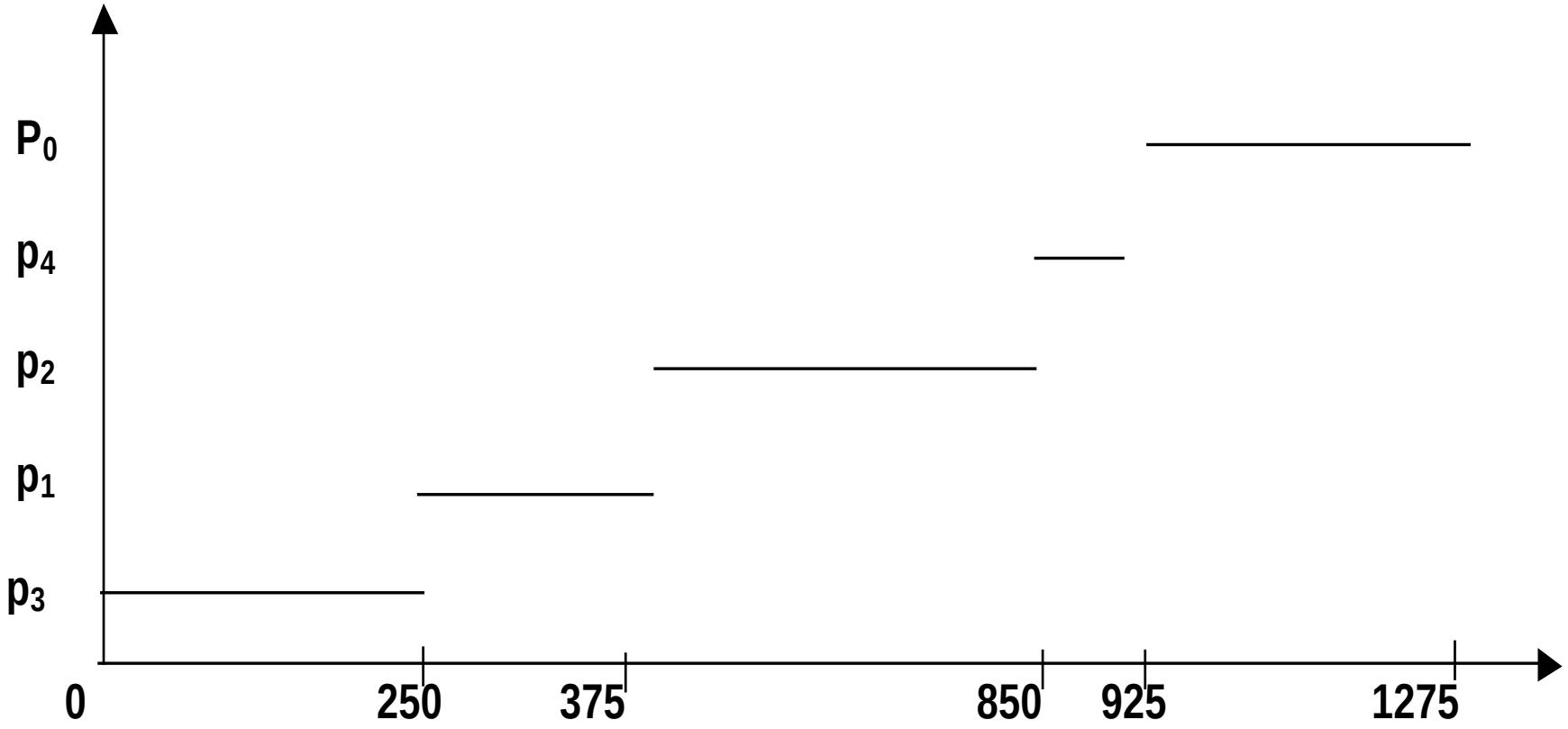
$$W = (0 + 75 + 200 + 450 + 800)/5 = 1525/5 = 305$$

- Se observă că valorile calculate ale factorilor de eficienţă sunt mult mai mici dacă se foloseşte această metodă faţă de cea anterioară.


p_2
p_0
p_3
p_1
p_4
0
75
200
450
800
1275

- **Algoritmul bazat pe priorităţi** este cel mai des folosit.
- În planificarea bazată pe priorităţi, proceselor le este alocată CPU pe baza unor priorităţi alocate extern. Aceste priorităţi sunt numere naturale, un proces având prioritate faţă de altul dacă numărul alocat are o valoare mai mică. Joburile se ordonează după aceste priorităţi, apoi se execută în această ordine. Se disting două tipuri de priorităţi: **interne** şi **externe**. Priorităţile interne sunt acordate pe baza unor particularităţi ale procesului, relative la mediului de calcul respectiv, cum ar fi de exemplu **timpul de execuţie**. Priorităţile externe reflectă **importanţa sarcinii** pe care procesul respectiv o execută, stabilită pe baza anumitor informaţii cum ar fi de, exemplu **numele utilizatorului**(cui aparţine procesul respectiv), **natura sarcinii**(cât de importantă este problema pe care o rezolvă) etc.
- Alegerea priorităţilor pentru procese este o problemă cheie, pentru ca această metodă de planificare să fie eficientă. Şi aici există pericolul apariţiei fenomenului de „înfometare”, adică procesele care au o prioritate mai slabă să nu fie executate. Pentru a se evita acest fenomen, se poate considera drept criteriu de acordare a priorităţilor timpul de când jobul respectiv se află în sistem.
- **Obs.** Toate celelalte strategii folosite, sunt cazuri particulare ale alg. bazat pe priorităţi.
- - Metoda FIFO: priorităţile job-urilor sunt calculate de administratorul cozii, pe baza valorii timpului când intră în lista READY.
- - Metod SJN: priorităţile job-urilor sunt calculate ţinâng cont de durata timpului de execuţie.

- **Exemplu.** Presupunem că avem *5* procese p_0, p_1, p_2, p_3, p_4 ale căror priorităţi sunt *5*, *2*, *3*, *1*, *4* iar timpii de execuţie sunt cei din exemplul anterior. Diagrama Gantt din figura urm. descrie ordinea execuţiei proceselor.



- Vom avea:

$$T(p_0) = \tau(p_0) + \tau(p_4) + \tau(p_2) + \tau(p_1) + \tau(p_3) = 350 + 75 + 475 + 125 + 250 = 1275$$

$$T(p_1) = \tau(p_1) + \tau(p_3) = 125 + 250 = 375$$

$$T(p_2) = \tau(p_2) + \tau(p_1) + \tau(p_3) = 475 + 125 + 250 = 850$$

$$T(p_3) = \tau(p_3) = 250$$

$$T(p_4) = \tau(p_4) + \tau(p_2) + \tau(p_1) + \tau(p_3) = 75 + 475 + 125 + 250 = 925$$

- Deci media timpului petrecut în sist. va fi

$$T = (1275 + 375 + 850 + 250 + 925)/5 = 735$$

- De asemenea, timpii de aşteptare se pot detemina într-un mod asemănător celui din exemplul anterior, aceştia fiind:
- $W(p_0)=925,\ W(p_1)=250,\ W(p_2)=375,\ W(p_3)=0,\ W(p_4)=850$
- Deci media timpului de aşteptare va fi
- $W(p_0)=(925+250+375+0+850)/5=480$

Strategii cu evacuare

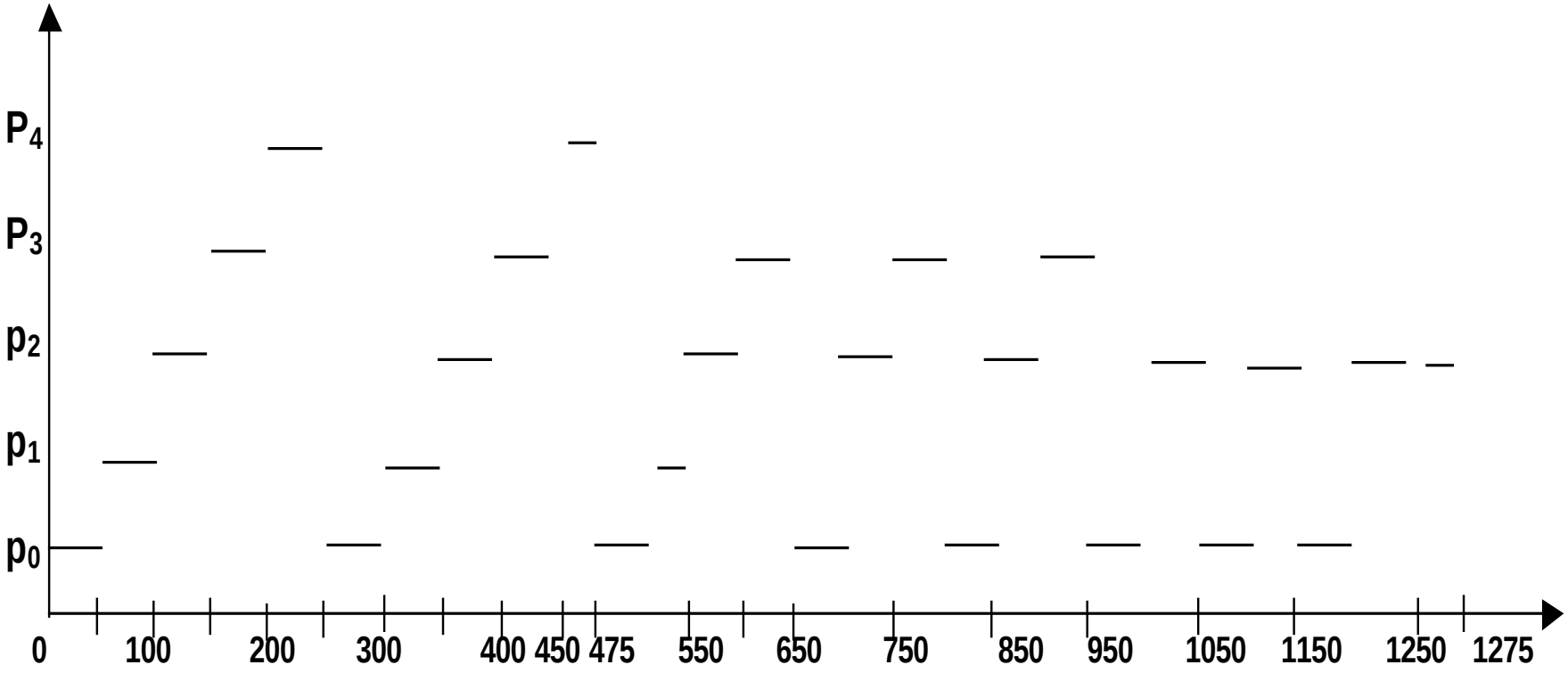
- În cazul algoritmilor cu evacuare;
- - CPU este alocată procesului cu prioritatea cea mai înaltă, dintre toate procesele care sunt în starea READY.
- - Toate procesele cu prioritatea mai mică cedează CPU procesului cu prioritatea cea mai înaltă, atunci când acesta o cere.
- - Oricând un proces intră în starea READY, el poate întrerupe imediat procesul servit de către CPU, dacă acesta are o prioritate mai slabă (planificatorul este apelat de fiecare dată când un proces intră în starea READY; de asemenea, el declanşează măsurătorul de intervale de timp, atunci când cuanta de timp a trecut).
- Strategiile cu evacuare sunt adesea folosite pentru a se asigura un răspuns rapid proceselor cu o prioriate înaltă sau pentru a se asigura o bună partajare a CPU între toate procesele.
- Există versiuni cu evacuare pentru unele dintre strategiile fără evacuare prezentate anterior. În cazul strategiei SJN, procesului cu cererea de timp cea mai mică îi este alocat CPU. Dacă p_i este în curs de execuţie şi intră în starea READY un alt proces p_j, atunci prin metoda SJN este necesară compararea lui $\tau(p_i)$ cu $\tau(p_j)$

- Acest lucru este posibil deoarece se cunoşte că timpul de servire al lui p_i este cel mai mic, dintre timpii tuturor proceselor aflate în starea READY. Dacă p_j cere un timp de execuţie mai mic decât timpul rămas până la terminarea execuţiei lui p_i, atunci lui p_j i se va aloca CPU, iar p_i va fi introdus în lista READY, cu $\tau(p_i)$ fiind timpul rămas până la terminarea execuţiei lui. Într-un mod similar, se procedează şi în cazul strategiei bazată pe împărţirea proceselor gata de execuţie în clase de priorităţi.
- **Observaţie**. În cazul algoritmilor fără evacuare, nu am considerat costul schimbării de context între procese, deoarece atunci am presupus că odată ce unui proces îi este alocată CPU, acesta va fi executat până la capăt. Această problemă capătă o importanţă deosebită în cazul strategiilor cu evacuare, datorită faptului că pe durata execuţiei unui proces acesta poate fi evacuat de mai multe ori, iar fiecare evacuare este însoţită de o schimbare de context, al cărei cost nu poate fi neglijat.

Planificarea circulară(RR-**R**ound **R**obin**)**

- **Planificarea circulară**(RR-**R**ound **R**obin**)** este poate cea mai utilizată dintre toţi algoritmii de planificare, în special de SO care lucrează în time-sharing. Principala ei caracteristică, este servirea echitabilă a tuturor proceselor care cer alocarea CPU.
- Să presupunem că în sistem avem *n* procese $p_0, .., p_{n-1}$, ordonate după indicele lor. Se defineşte o cuantă de timp, dependentă de sistem şi pe durata unei cuante se alocă procesorul unui proces.
- Coada READY a proceselor este tratată circular; mai întâi este servit p_0, apoi p_1, ş.a.m.d p_{n-1}, după care este servit din nou p_0.
- Dacă procesorul termină de servit un proces înainte de expirarea cuantei de timp alocate, o nouă cuantă de timp este definită şi aceasta este alocată procesului următor.
- Când se termină execuţia unui proces, el este scos din lista READY.
- Când un nou proces este introdus în lista READY, este inserat ultimul în această coadă circulară. De fiecare dată când apare un astfel de eveniment, planificatorul este apelat imediat şi, eventual se defineşte o nouă cuantă de timp şi se selectează un nou proces.
- În cazul acestui algoritm, trebuie să luăm în consideraţie efectul schimbării de context între procese. Fie C timpul necesr efectuării acestei operaţii. Dacă fiecăruia dintre cele *n* procese îi este alocat *q* unităţi de timp CPU, atunci timpul total de servire al celor *n* procese va fi $n(q+C)$.

- **Exemplu.** Să considerăm procesele p_0, p_1, p_2, p_3, p_4 care au timpii de execuţie definiţi în exemplele anterioare, iar cuanta de timp este de *50* de unităţi şi presupunem că se neglijează timpul necesar schimbării de context. În figura urm. este prezentată ordinea servirii proceselor şi momentele terminării execuţiei lor.



- Din diagrama din figura ant. putem deduce:
- Timpii petrecuţi în sistem de fiecare proces: $T(p_0)=1100$, $T(p_1)=550$, $T(p_2)=1275$, $T(p_3)=950$, $T(p_4)=475$ şi media timpului petrecut în sistem de un proces $T=(1100+550+1275+950+475)/5=870$.
- Timpii de aşteptare până la prima servire: $W(p_0)=0$, $W(p_1)=50$, $W(p_2)=100$, $W(p_3)=150$, $W(p_4)=200$ şi media acestor timpi $W=(0+50+100+150+200)/5=100$.
- Dacă şi timpul de comutare între procese, ca fiind de 10 unităţi de timp, obţinem: $T(p_0)=1320$, $T(p_1)=660$, $T(p_2)=1535$, $T(p_3)=1140$, $T(p_4)=565$, $T=(1320+660+1535+1140+565)/5=1044$, $W(p_0)=0$, $W(p_1)=60$, $W(p_2)=120$, $W(p_3)=180$, $W(p_4)=220$ şi $W=(0+60+120+180+240)/5=120$.
- **Metoda cozilor pe mai multe niveluri** este o extensie a planificării bazate pe priorităţi, care presupune că toate procesele care au aceeaşi prioritate sunt plasate în aceeaşi coadă. Între cozi, planificatorul alocă CPU folosind o anumită strategie, iar în interiorul cozii o altă strategie. De exemplu, la un SO de tip mixt, interactiv şi serial, pot fi create *5* cozi distincte: **taskuri sistem**, **lucrări interactive**, **lucrări în care se editează texte**, **lucrări seriale obişnuite**, **lucrări seriale ale studenţilor** etc.