

# Sisteme de Operare

## Gestiunea proceselor – partea II

**Cristian Vidrașcu**

<http://www.info.uaic.ro/~vidrascu>

# Cursul precedent

- Conceptul de proces
- Stările procesului
- Relații între procese
- Procese concurente
- Planificarea proceselor
  - Obiective
  - Cozi de planificare
  - Planificatoare

- Planificarea proceselor (continuare)
  - Structura planificării
  - Schimbarea contextului
  - Priorități
  - Algoritmi de planificare:  
FCFS, SJF, Priorități, RR, ș.a.

# Structura planificării /1

- Structura planificării

- Deciziile de planificare a CPU se iau în următoarele situații:

1. când un proces trece din starea running în starea waiting  
(e.g. cerere I/O, sau apel wait)
2. când un proces trece din starea running în starea ready  
(e.g. când apare o întrerupere hardware de ceas ce marchează sfârșitul unei cuante de timp procesor)
3. când un proces trece din starea waiting în starea ready  
(e.g. terminarea unei operații I/O)
4. când un proces se termină

# Structura planificării /2

- Structura planificării

- Pentru situațiile 1. și 4., un nou proces (dacă există vreunul în starea ready) trebuie să fie selectat pentru execuție.
- Când planificarea se face numai datorită situațiilor 1. și 4., schema de planificare este numită **ne-preemptivă**; altfel, ea este **preemptivă**.
- O politică de planificare este numită **preemptivă** dacă, o dată ce unui proces i s-a dat CPU-ul, acesta poate mai târziu să-i fie luat.  
Și este **ne-preemptivă** dacă CPU-ul nu mai poate fi luat procesului ce-l deține, i.e. fiecare proces rulează până la terminare sau la efectuarea unei cereri I/O, sau apel wait.

# Structura planificării /3

- Structura planificării
  - O politică ne-preemptivă implică mai puțină încărcătură (*overhead*) a sistemului și face ca timpul total de la startul execuției unui program și până la terminarea lui să fie mai ușor de anticipat.
  - Schemele preemptive sunt importante în sistemele în timp real și în cele cu timp partajat, dar acestea implică schimbarea frecventă a proceselor pe CPU (*process switching*), ceea ce poate determina o încărcare suplimentară semnificativă a sistemului.

# Structura planificării /4

- **Preempție**

- Politicile de planificare pot fi *preemptive* sau *ne-preemptive*.  
*Preempție*: planificatorul poate forța un proces să renunțe la procesor înainte ca procesul să se blocheze (i.e. să inițieze o operație I/O), să renunțe singur la CPU, sau să se termine.

- **Cuantificarea timpului CPU** (*timeslicing*) previne monopolizarea CPU-ului de către vreun proces

- Planificatorul alege un proces ready și-l execută o *cuantă* de timp.
- Un proces ce se execută mai mult decât cuanta sa de timp, este forțat să renunțe la CPU de către codul planificatorului rulat prin *handler*-ul întreruperii hardware de ceas.

- In politicile de planificare pe bază de **priorități** se folosește preempția pentru a onora prioritățile

- Procesul curent running este preemptat dacă un proces cu o prioritate mai mare intră în starea ready.

# Schimbarea contextului

- **Schimbarea contextului** (*context switch*)
  - Comutarea CPU-ului către alt proces necesită salvarea stării vechiului proces și încărcarea stării salvate a noului proces ce urmează să se execute
  - Timpul necesar pentru schimbarea contextului constituie o încărcare suplimentară a sistemului (de ordinul 1-100 microsecunde), dar depinde foarte mult de suportul oferit de hardware



- **Prioritate**

- Anumite obiective pot fi îndeplinite prin încorporarea în disciplina de planificare de bază (gen round-robin) a unei noțiuni de *prioritate* a proceselor.
- Fiecare proces din coada ready are asociată o anumită valoare a priorității; planificatorul favorizează procesele cu valori mai ridicate ale priorității.

# Priorități /2

- Valori *externe* ale priorității
  - sunt impuse sistemului din afara sa
  - reflectă preferințe externe pentru anumiți utilizatori sau joburi  
 (“Toate joburile sunt egale, dar unele sunt mai egale decât altele...”)
  - exemplu: primitiva Unix **nice()** micșorează prioritatea unui job
  - exemplu: joburile urgente într-un sistem de control în timp real
- Manipularea priorităților
  - **extern** – în funcție de rangul utilizatorului, ș.a.
  - **intern** – planificatorul calculează dinamic prioritățile și le utilizează pentru gestiunea cozilor de planificare

Sistemul ajustează intern valorile priorităților printr-o tehnică implementată în planificator.

# Priorități /3

Prioritățile trebuie manevrate cu grijă atunci când există dependențe între procese cu priorități diferite.

- Un proces cu prioritatea  $P$  ar trebui să nu împiedice niciodată progresul unui proces cu prioritatea  $Q > P$ .

O astfel de situație se numește *inversiunea priorității* și trebuie să se evite apariția sa.

- Soluția cea mai simplă constă într-o *moștenire a priorității*:

Când un proces cu prioritatea  $Q$  așteaptă o anumită resursă, deținătorul ei (cu prioritatea  $P$ ) moștenește temporar prioritatea  $Q$  dacă  $Q > P$ .

Moștenirea s-ar putea să fie necesară și atunci când procesele se coordonează prin IPC (Inter-Process Communication).

- Moștenirea este utilă și în alte situații, spre exemplu, pentru a îndeplini anumite termene limită.

# Planificarea proceselor

## ➤ Algoritmi de planificare

### – Criterii:

- Gradul de utilizare a CPU-ului (% timp non-idle; 40%-90%)
- Rata de servire (numărul de procese/unitatea de timp)
- Timpul turnaround (intervalul scurs între momentul submiterii și cel al terminării unui proces; timpul de viață)
- Timpul de așteptare (timpul petrecut în coada ready)
- Timpul de răspuns (timpul scurs între emiterea unei comenzi de către utilizator și producerea primului răspuns la acea comandă)

### – Scopuri:

- Maximizarea utilizării CPU și a ratei de servire
- Minimizarea timpilor turnaround, de așteptare și de răspuns

# Algoritmi de planificare /1

## ➤ **Algoritmi de planificare**

- First-Come, First-Served (FCFS)
- Shortest-Job-First (SJF)
- Planificarea cu priorități
- Round-Robin (RR)
- Planificarea cu cozi pe nivele multiple
- Planificarea în timp real
- Planificarea cu procesoare multiple

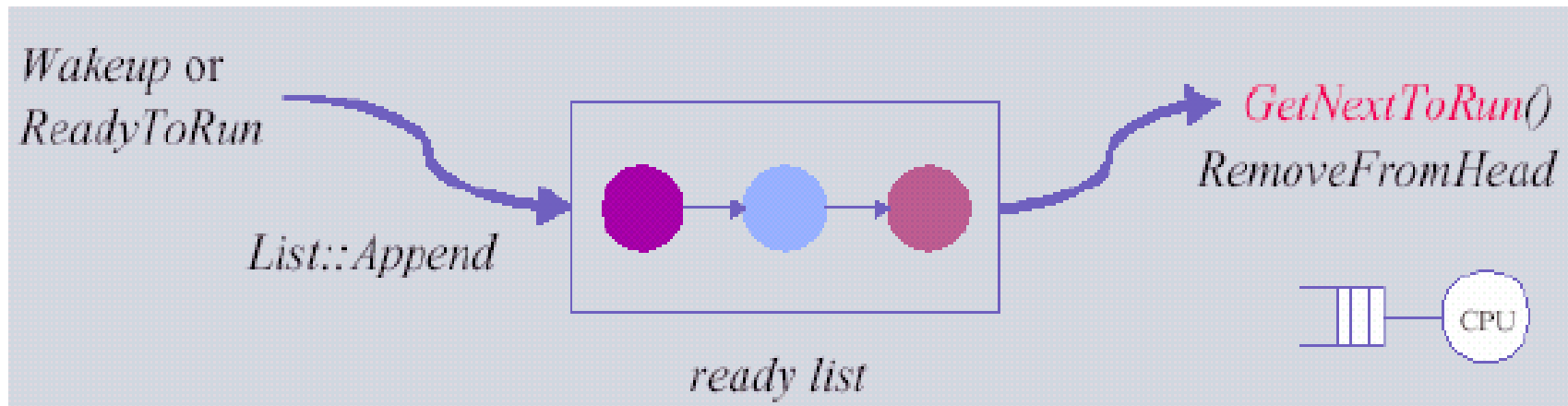
# Algoritmi de planificare /2

- **First-Come, First-Served (FCFS)**
  - procesul care solicită primul să i se acorde timp CPU, este primul căruia i se va aloca CPU-ul
  - implementare: o simplă structură FIFO
  - algoritmul este simplu de scris și de înțeles
  - alg. de planificare FCFS este ne-preemptiv (deci nu poate fi utilizat pentru medii interactive)
  - procesele lungi sunt favorizate de politica de planificare FCFS, iar cele scurte sunt defavorizate

# Algoritmi de planificare /3

- **First-Come, First-Served (FCFS)**

- **rata de servire** – alg. FCFS este la fel de bun ca orice altă politică de planificare ne-preemptivă ...  
... dacă CPU-ul ar fi singura resursă planificabilă din sistem
- **echitate** – alg. FCFS este intuitiv echitabil
- **timpul de răspuns** – procesele lungi le țin pe toate celelalte în așteptare



# Algoritmi de planificare /4

## • First-Come, First-Served (FCFS)

Scenariu:

Ipoteză :  
doar procese  
CPU-intensive  
(fără nici o op. I/O)

| Procese | Momentul sosirii | Timpul de serviciu solicitat |
|---------|------------------|------------------------------|
| A       | 0                | 3                            |
| B       | 1                | 5                            |
| C       | 3                | 2                            |
| D       | 9                | 5                            |
| E       | 12               | 5                            |

|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| A  | A  | A  | B  | B  | B  | B  | B  | C  | C  | D  | D  | D  | D  | D  | E  | E  | E  | E  | E  |



# Algoritmi de planificare /5

- **First-Come, First-Served (FCFS)**

Rezultatele  
planificării:

| Proces | Sosire | Serviciu | Start | Finish | T | W | P   |
|--------|--------|----------|-------|--------|---|---|-----|
| A      | 0      | 3        | 0     | 3      | 3 | 0 | 1.0 |
| B      | 1      | 5        | 3     | 8      | 7 | 2 | 1.4 |
| C      | 3      | 2        | 8     | 10     | 7 | 5 | 3.5 |
| D      | 9      | 5        | 10    | 15     | 6 | 1 | 1.2 |
| E      | 12     | 5        | 15    | 20     | 8 | 3 | 1.6 |

**t** = timpul de execuție (timpul serviciu)

**T** = timpul de răspuns (= timp finish – timp sosire)

**W** = timpul de așteptare (=  $T - t$ )

**P** = rata de penalitate =  $T/t$  ; **R** = rata de răspuns =  $t/T$

# Algoritmi de planificare /6

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

- **Ideea:** scoaterea rapidă din sistem a proceselor scurte pentru a minimiza numărul de procese aflate în așteptare cât timp rulează un proces lung
- Intuitiv: procesele cele mai lungi dăunează cel mai mult pentru timpii de așteptare ai competitorilor lor
- SJF este **optimal** (lucru demonstrabil matematic), în sensul că produce cel mai mic timp mediu de așteptare pentru o mulțime dată de procese
- Este nevoie de **anticiparea** timpilor de serviciu CPU

# Algoritmi de planificare /7

## • Shortest-Job-First (SJF)

- Planificarea SJF poate fi ne-preemptivă sau preemptivă
- Planificarea SJF preemptivă este numită planificare **shortest-remaining-time-first**
- SJF favorizează procesele interactive, ce necesită răspuns rapid și care interacționează cu utilizatorul în mod repetat
- SJF favorizează procesele ce produc rafale (*bursts*) I/O – care se blochează curând, țin perifericele ocupate, eliberând astfel CPU-ul
- Atenția este îndreptată spre o măsură *medie* a performanței, unele procese lungi pot fi înfometate în cazul unei încărcări masive a sistemului sau a unui flux constant de noi procese scurte ce intră în sistem

# Algoritmi de planificare /8

## • Shortest-Job-First (SJF)

- Sacrifică echitatea pentru a micșora timpul mediu de răspuns
- Planificarea SJF pură este impracticabilă: planificatorul nu poate anticipa durata unui proces
- Totuși, SJF are valoare în sistemele reale:
  - Multe aplicații execută o secvență de rafale CPU scurte cu operații I/O între acestea
  - E.g., joburile *interactive* se blochează în mod repetat pentru a accepta input din partea utilizatorului

Scop: furnizarea celui mai bun timp de răspuns pentru utilizator

- E.g., joburile pot trece prin perioade de activitate I/O intensivă  
Scop: cererea următoarei operații I/O cât mai repede posibil pentru a ține perifericele ocupate și a furniza cea mai bună rată de servire pe ansamblu
- Folosirea *priorității interne adaptive* pentru a încorpora SJF în RR  
Strategia meteorologilor: previziunea viitorului apropiat pe baza trecutului recent

# Algoritmi de planificare /9

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

|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| A  | A  | A  | C  | C  | B  | B  | B  | B  | B  | D  | D  | D  | D  | D  | E  | E  | E  | E  | E  |

| Proces | Sosire | Serviciu | Start | Finish | T | W | P   |
|--------|--------|----------|-------|--------|---|---|-----|
| A      | 0      | 3        | 0     | 3      | 3 | 0 | 1.0 |
| B      | 1      | 5        | 5     | 10     | 9 | 4 | 1.8 |
| C      | 3      | 2        | 3     | 5      | 2 | 0 | 1.0 |
| D      | 9      | 5        | 10    | 15     | 6 | 1 | 1.2 |
| E      | 12     | 5        | 15    | 20     | 8 | 3 | 1.6 |

Rezultatele  
planificării

# Algoritmi de planificare /10

- **Planificarea cu priorități**

- Fiecare proces are asociată o prioritate, iar CPU-ul este alocat procesului ready cu prioritatea cea mai mare
- Procesele cu priorități egale sunt planificate în ordinea FCFS
- Valorile priorităților depind de implementarea S.O. (numerele mici pot semnifica prioritati mari, sau invers)
- Observație: SJF poate fi privit ca un algoritm cu priorități, unde prioritatea ( $p$ ) este inversul duratei următoarei rafale CPU anticipate ( $c$ ) :  $p = 1/c$

- **Planificarea cu priorități**

- Poate fi preemptivă sau ne-preemptivă

- Problemă:

- Blocarea nelimitată sau înfometarea (starvation)*  
proceselor cu priorități mici

- Soluție:

- Imbătrânirea (aging)* : creșterea graduală a priorității  
proceselor care așteaptă în sistem pentru o perioadă  
mare de timp

# Algoritmi de planificare /12

## • Planificarea Round-Robin (RR)

- Fiecărui proces  $i$  se oferă serviciul CPU pentru o perioadă scurtă de timp (numită *cuantă* sau *felie de timp*), după care, doar dacă nu a fost terminat, revine la sfârșitul cozii ready și așteaptă să-i vină din nou rândul la CPU
- Această perioadă este adesea 100 ms (sau chiar mai puțin, 10ms)
- A fost proiectată special pentru sistemele cu timp partajat
- Coadă ready este tratată ca o coadă circulară
- Comutarea CPU-ului de la un proces la altul (schimbarea contextului) implică o încărcare suplimentară considerabilă a sistemului



# Algoritmi de planificare /13

- **Planificarea Round-Robin (RR)**

- *Timpul de răspuns*

RR reduce timpul de răspuns pentru procesele scurte

- Pentru o încărcare dată a sistemului, timpul de răspuns al unui proces este direct proporțional cu durata sa

- *Echitate*

RR reduce variația în timpii de așteptare

- Dar: RR forțează procesele să aștepte pentru alte procese intrate mai târziu în sistem

- *Throughput (rata de servire)*

RR impune o încărcare suplimentară a sistemului pentru schimbarea contextului, ceea ce dăunează ratei

# Algoritmi de planificare /14

- **Planificarea Round-Robin (RR)**

- Pe un sistem *multiprocesor*, RR poate îmbunătăți rata de servire sub o încărcare ușoară (i.e. număr mic de joburi)
- *Scenariu*: 3 fripturi trebuie gătite timp de 5 minute pe fiecare față, dar pe plită este loc doar pentru 2 fripturi
  - 30 minute timpul secvențial necesar pentru gătit: fripturile 1,2,3 cu fețele A,B
- **FCFS**: fripturile 1,2 timp de 10 minute, apoi friptura 3 timp de 10 minute
  - Se termină în 20 minute, cu gradul de utilizare a plitei: 75%
- **RR**: ordinea 1A și 2A ... flip ... 1B și 3A ... flip ... 2B și 3B
  - Se termină în 3 cuante (15 minute), cu o utilizare de 100% a plitei

# Algoritmi de planificare /15

## • Planificarea Round-Robin (RR)

Primul scenariu (o cuantă = 1 unitate de timp)

|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| A  | B  | A  | B  | C  | A  | B  | C  | B  | D  | B  | D  | E  | D  | E  | D  | E  | D  | E  | E  |

| Proces | Sosire | Serviciu | Start | Finish | T  | W | P   |
|--------|--------|----------|-------|--------|----|---|-----|
| A      | 0      | 3        | 0     | 6      | 6  | 3 | 2.0 |
| B      | 1      | 5        | 1     | 11     | 10 | 5 | 2.0 |
| C      | 3      | 2        | 4     | 8      | 5  | 3 | 2.5 |
| D      | 9      | 5        | 9     | 18     | 9  | 4 | 1.8 |
| E      | 12     | 5        | 12    | 20     | 8  | 3 | 1.6 |

Rezultatele  
planificării

# Algoritmi de planificare /16

## • Planificarea Round-Robin (RR)

Al doilea scenariu (o cuantă = 4 unități de timp)

|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| A  | A  | A  | B  | B  | B  | B  | C  | C  | B  | D  | D  | D  | D  | E  | E  | E  | E  | D  | E  |

| Proces | Sosire | Serviciu | Start | Finish | T  | W | P   |
|--------|--------|----------|-------|--------|----|---|-----|
| A      | 0      | 3        | 0     | 3      | 3  | 0 | 1.0 |
| B      | 1      | 5        | 3     | 10     | 9  | 4 | 1.8 |
| C      | 3      | 2        | 7     | 9      | 6  | 4 | 3.0 |
| D      | 9      | 5        | 10    | 19     | 10 | 5 | 2.0 |
| E      | 12     | 5        | 14    | 20     | 8  | 3 | 1.6 |

Rezultatele  
planificării

# Algoritmi de planificare /17

## • Planificarea Round-Robin (RR)

- Problemă: Cât ar trebui să fie cuanta de timp ?
- Dacă e prea mare, pur și simplu RR devine FIFO
- Dacă e prea mică, CPU-ul este folosit (aproape) în întregime doar de către rutina dispecer pentru selecție și comutarea contextului, deci joburile utilizatorilor nu mai apucă să fie executate
- Durata ideală a cuantei de timp depinde de mulți factori
- O bună regulă generală este: acea perioadă de timp în care marea majoritate a utilizatorilor interactivi pot fi satisfăcuți ( $10 \div 100$  ms)
- Durata cuantei este adesea schimbată de două ori pe zi (este mai lungă noaptea, când rulează joburi de tip batch)

# Algoritmi de planificare /18

- **Planificarea cu cozi pe nivele multiple**

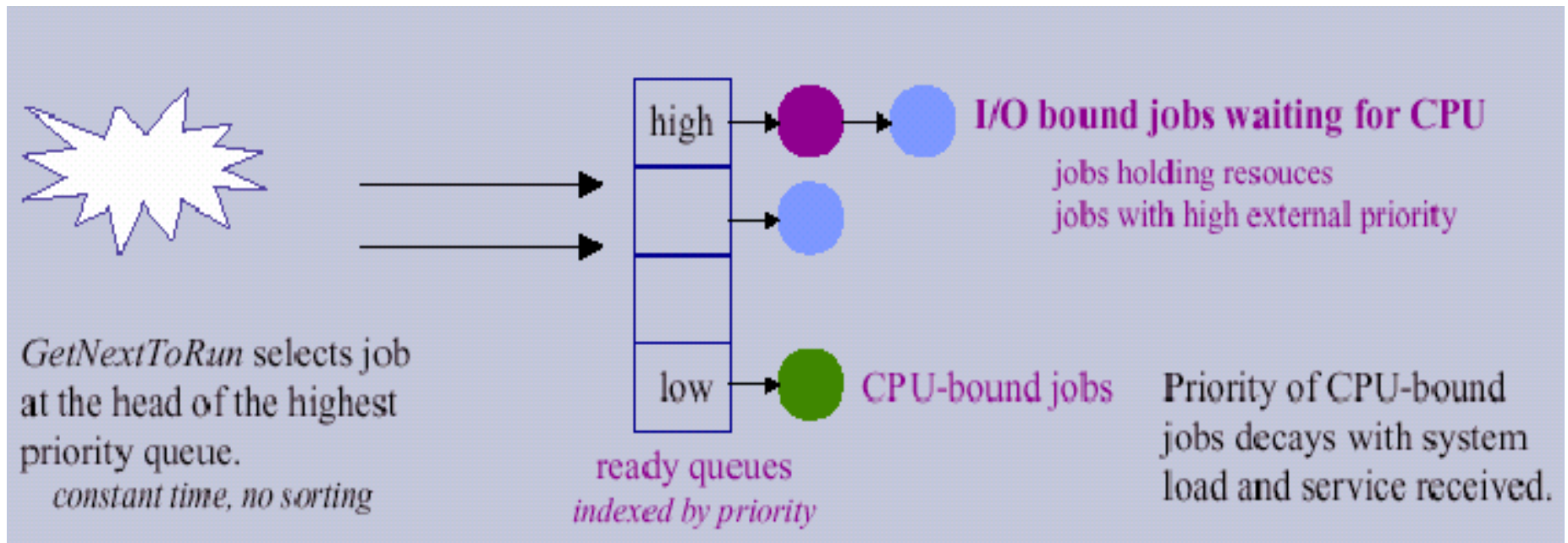
- Divizarea cozii ready în mai multe (sub-)cozi ready separate

E.g., coada proceselor **foreground (interactive)** și coada proceselor **background (batch)**

- Procesele sunt asignate permanent uneia dintre cozi, în general pe baza unei proprietăți a procesului (dimensiunea memoriei, tipul procesului, etc.)
- Fiecare coadă are propriul algoritm de planificare
- Trebuie să existe o planificare între cozi (e.g., poate fi utilizată o planificare preemptivă cu priorități fixe)

# Algoritmi de planificare /19

- Planificarea cu cozi pe nivele multiple

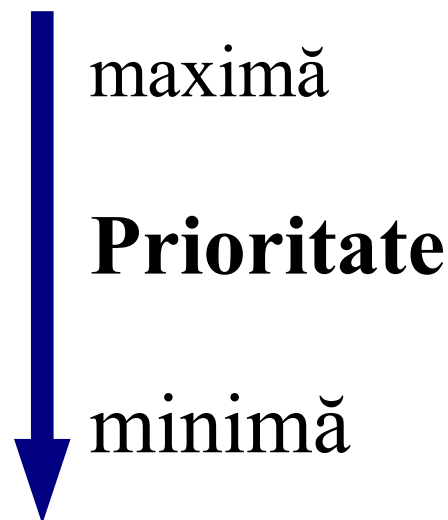


# Algoritmi de planificare /20

- **Planificarea cu cozi pe nivele multiple**

- Exemplu:

- Procese de sistem
- Procese interactive
- Procese de editare de texte
- Procese batch



- Fiecare coadă are întâietate absolută asupra cozilor cu priorități mai joase
- Altă soluție: de împărțit timpul în cuante (*time slice*) între cozi



# Algoritmi de planificare /21

- **Planificarea cu cozi pe nivele multiple, cu feedback**
  - Se permite unui proces să migreze între cozi
  - Parametri:
    - Numărul de cozi
    - Algoritmul de planificare pentru fiecare coadă
    - Metoda utilizată pentru a decide când un proces să fie avansat într-o coadă cu prioritate mai mare
    - Metoda utilizată pentru a decide când un proces să fie degradat într-o coadă cu prioritate mai mică
    - Metoda utilizată pentru a decide în ce coadă va intra inițial un proces (i.e., atunci când acesta este lansat în execuție)

# Algoritmi de planificare /22

- **Planificarea în timp real**

- Planificatoarele în timp real trebuie să suporte execuții regulate, periodice, ale taskurilor (e.g., flux media continuu)

- *Rezervări timp CPU*

- “Am nevoie să execut X din fiecare Y unități de timp.”

- Planificatorul exercită *controlul admiterii* la momentul rezervării: aplicația trebuie să gestioneze eșecul unei cereri de rezervare

- *Constrângeri temporale*

- “Rulează această aplicație înainte de termenul meu limită: momentul T.”

# Algoritmi de planificare /23

- **Planificarea cu procesoare multiple**  
(multiprocesare)
  - Probleme:
    - Tipul procesoarelor:  
sisteme omogene vs. sisteme heterogene
    - Tipul cozilor de planificare:  
coada ready comună vs. câte o coadă ready separată pentru fiecare procesor
  - Sisteme SMP: toate procesele intră într-o coadă și sunt planificate pe oricare dintre procesoarele disponibile
  - Planificare master-slave (multiprocesare asimetrica)

- **Bibliografie obligatorie**  
capitolele despre *gestiunea proceselor* din
  - Silberschatz : “*Operating System Concepts*”  
(cap.5 din [OSCE8])
  - sau
  - Tanenbaum : “*Modern Operating Systems*”  
(a patra parte a cap.2 din [MOS3])

- Planificarea proceselor (continuare)
  - Structura planificării
  - Schimbarea contextului
  - Priorități
  - Algoritmi de planificare:  
FCFS, SJF, Priorități, RR, ș.a.

Întrebări ?