

Sisteme de Operare

Gestiunea proceselor partea a II-a

Cristian Vidrașcu

<https://profs.info.uaic.ro/~vidrascu>

Cursul precedent

- Conceptul de proces (/fir de execuție)
- Stările procesului (/unui fir de execuție)
- Relații între procese
- Procese concurente
- Planificarea proceselor (/firelor de execuție)
 - Obiective
 - Cozi de planificare
 - Planificatoare
 - Schimbarea contextului
 - Priorități

- Planificarea proceselor (continuare)
 - Structura planificării
 - Algoritmi de planificare:
 - FCFS
 - SJF și SRTF
 - Planificarea cu priorități
 - RR
 - ș.a.

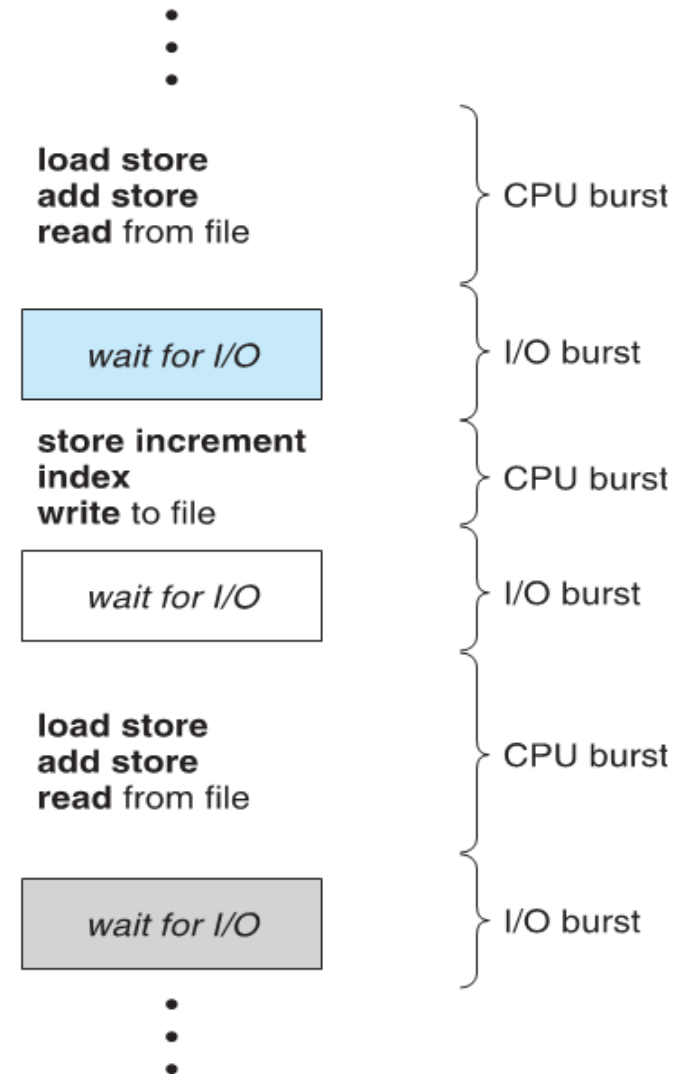
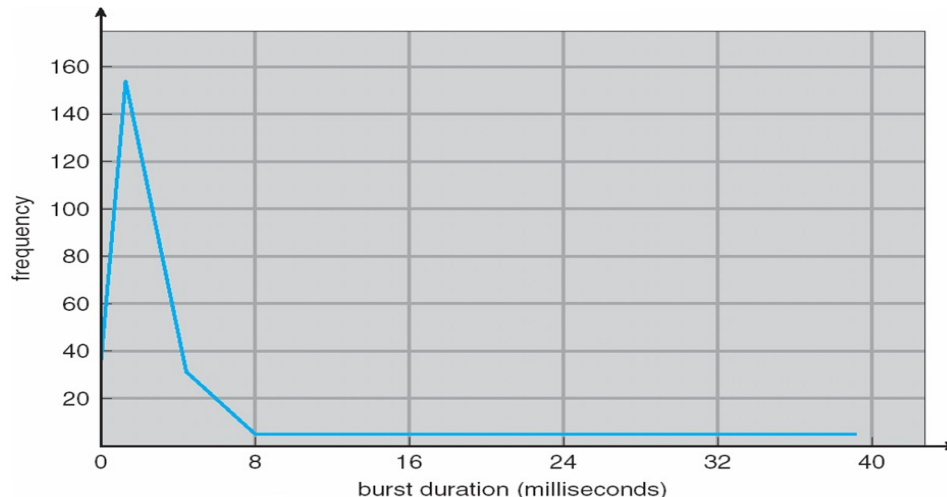
Structura planificării /1

- Concepte de bază

Execuția unui proces/thread constă într-o secvență de calcule executate pe CPU și de operații de I/O:

rafale CPU urmate de rafale I/O

Distribuția statistică *uzuală* a duratelor rafalelor CPU:



Structura planificării /2

- Structura planificării

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

1. când un proces/thread trece din starea running în starea waiting (e.g. cerere I/O, sau apel wait)
2. când un proces/thread 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/thread trece din starea waiting în starea ready (e.g. terminarea unei operații I/O, sau apel wait)
4. când execuția unui proces/thread se termină (normal sau forțat)

Structura planificării /3

- Structura planificării

- Pentru situațiile 1. și 4., un nou proces/thread (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/thread i s-a alocat CPU-ul, acesta poate mai târziu să-i fie luat (în mod forțat).
Și este **ne-preemptivă** dacă CPU-ul nu mai poate fi luat de la cel care-l deține, i.e. fiecare proces/thread rulează până la terminare sau până la efectuarea unei cereri I/O (sau apel wait).

Structura planificării /4

- **Preemptie**

- Politicile de planificare pot fi *preemptive* sau *ne-preemptive*.

- Preemptie*: 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.

- În politicile de planificare pe bază de **priorități** se folosește preemptia pentru a onora prioritățile

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

Structura planificării /5

- Observații:

- 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. Astfel de politici s-au utilizat îndeosebi în sistemele seriale (cu batch job-uri).
- Schemele preemptive implică schimbarea frecventă a proceselor pe CPU (prin *context switching*), ceea ce poate determina o încărcare suplimentară semnificativă a sistemului. Ele sunt utilizate în sistemele interactive (cu timp partajat), precum și în cele în timp real, pentru a putea satisface constrângerile legate de timpul de răspuns.

Planificarea proceselor

➤ Algoritmi de planificare

– Criterii de optimizare utilizate:

- **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) și SRTF
- Planificarea cu priorități, preemptivă și ne-preemptivă
- Round-Robin (RR)
- Planificarea cu cozi ready pe nivele multiple, simplă și cu feedback
- Planificarea în timp real
- Planificarea cu procesoare multiple

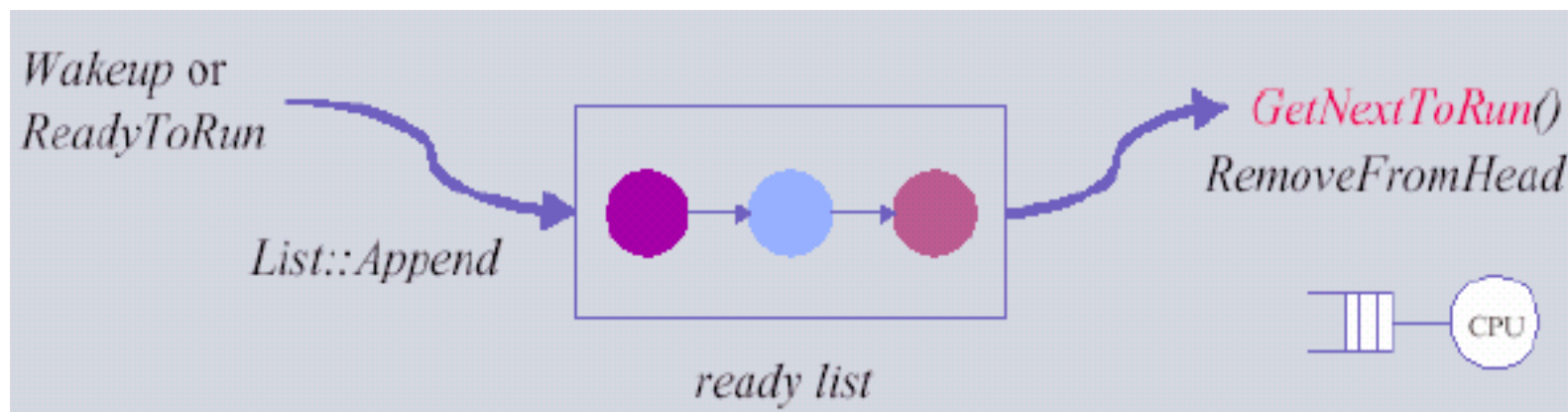
Algoritmi de planificare /2

- **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)

Scenariul #1:

(un exemplu
doar cu 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

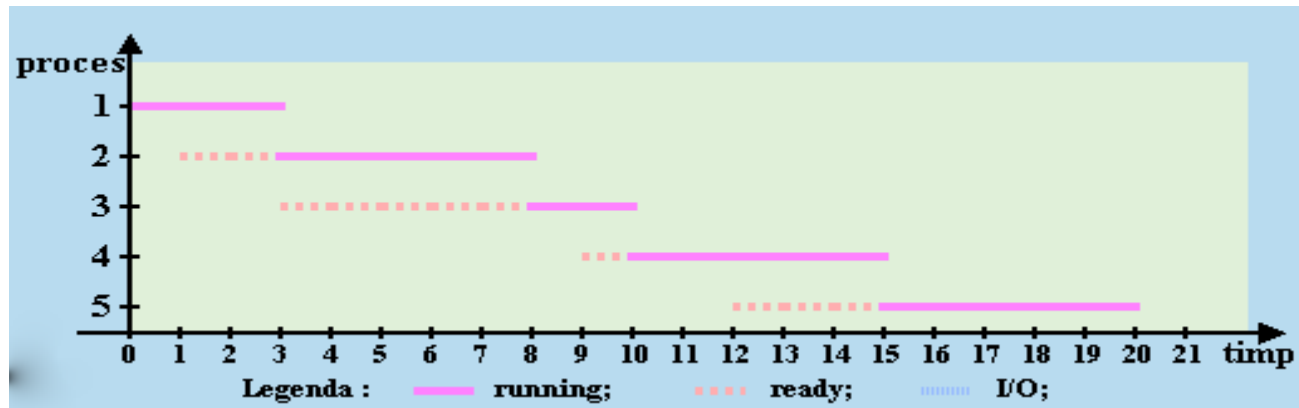
Ipoteză de lucru:

(pentru următoarele exemple)

Momentul sosirii
este “Sosire – ϵ ”

(Opțiunea alternativă
ar fi “Sosire + ϵ ”)

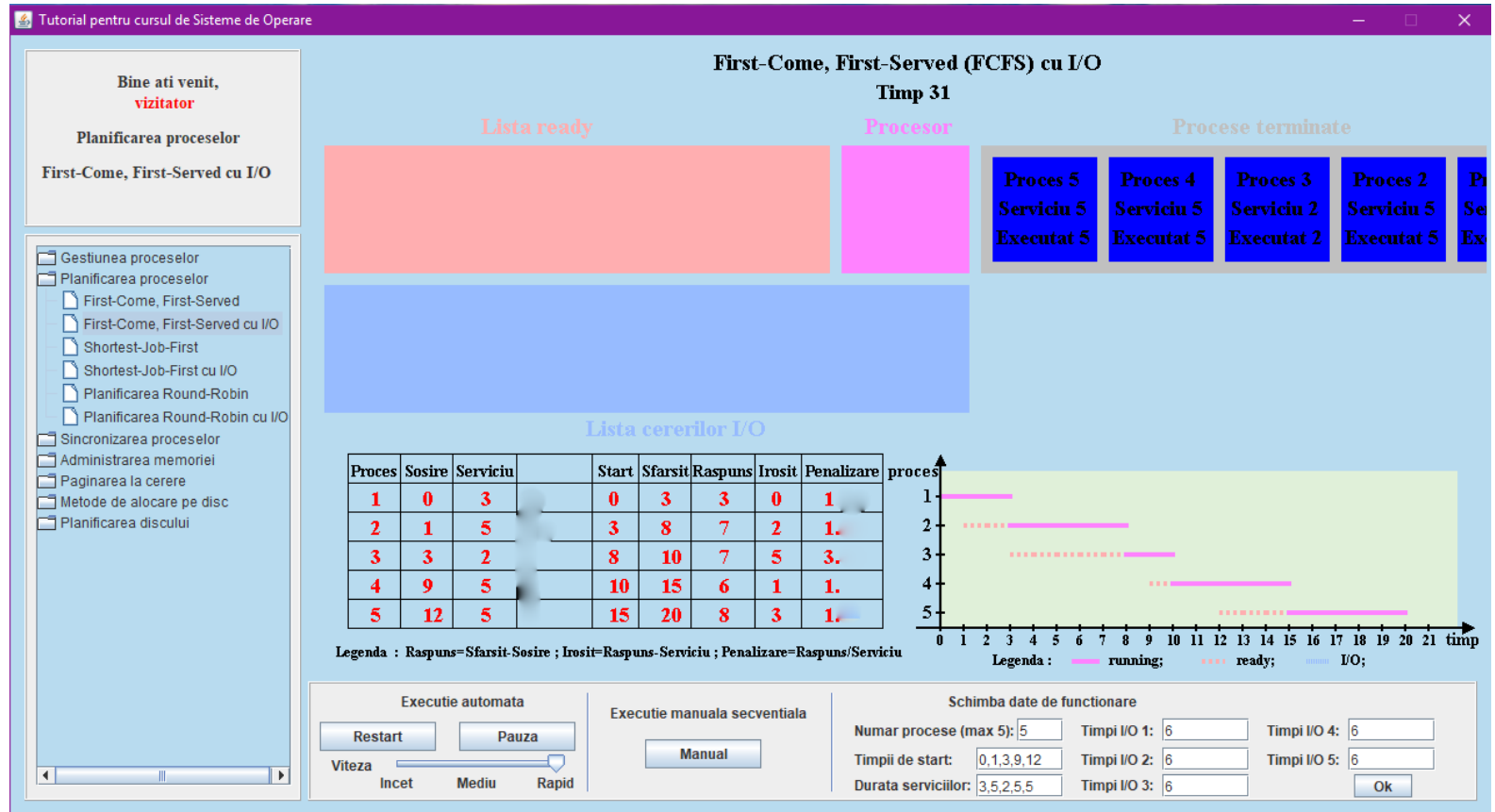
Diagrama timp
a planificării
(pe 1 CPU) :



Algoritmi de planificare /5

• FCFS

Scenariul #1 (fără I/O)



Sumarul planificării:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
A	A	A	B	B	B	B	B	C	C	D	D	D	D	D	E	E	E	E	E	

Algoritmi de planificare /6

• FCFS

Scenariul #1

Rezultatele
planificării
FCFS:

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

Notatii:

Start = momentul când devine *running*

Finish = momentul când se termină de executat

t = timpul de execuție propriu-zisă (i.e. timpul de serviciu)

T = timpul de viață (= moment finish – moment sosire)

W = timpul de așteptare (în coada ready) (= $T - t$)

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

Algoritmi de planificare /7

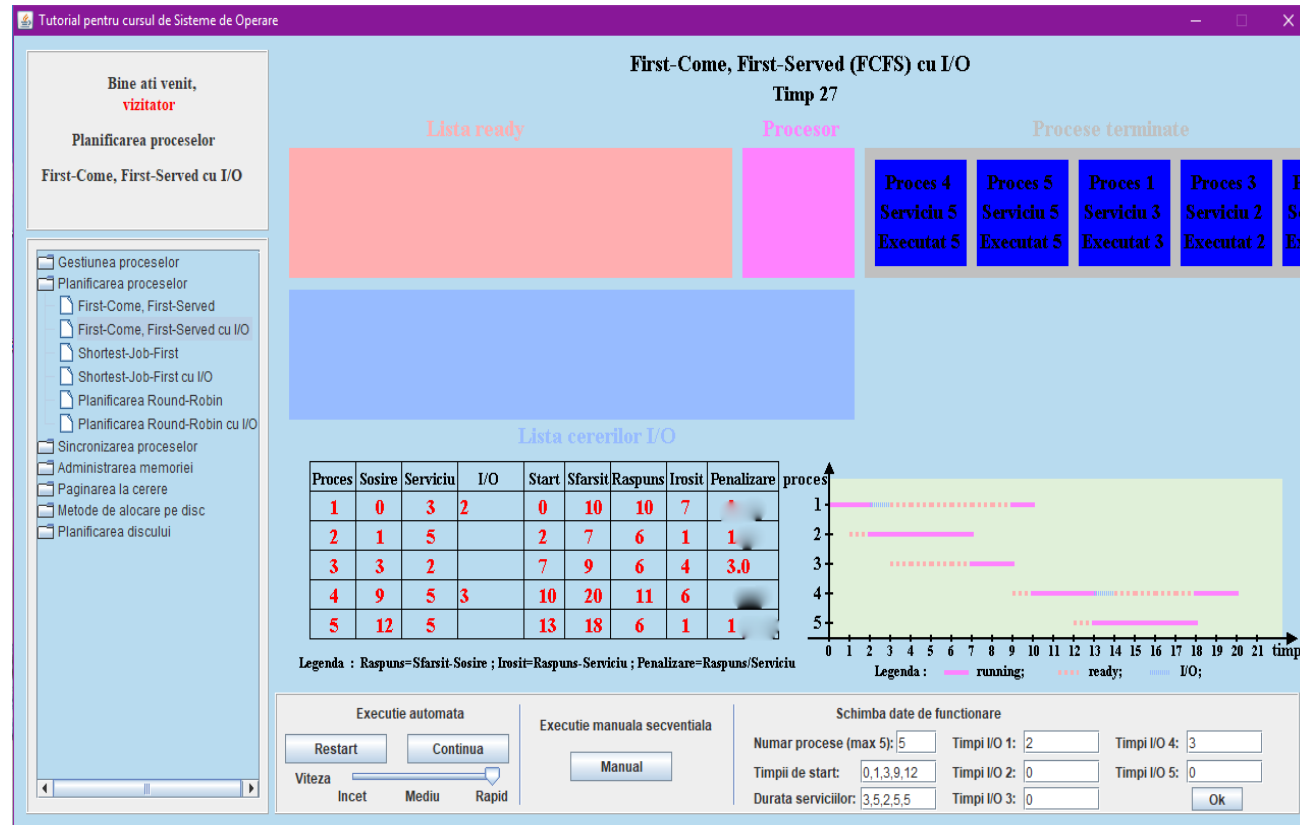
• FCFS

Scenariul #2 :

un exemplu în care (unele)
procese execută și operații I/O

(pentru simplitate, în cazul acestora
în coloana Serviciu se vor specifica
atât duratele rafalelor CPU, cu negru,
cât și duratele operațiilor I/O, cu roșu)

Proces	Sosire	Serviciu
A	0	2; 1 ;1
B	1	5
C	3	2
D	9	3; 1 ;2
E	12	5

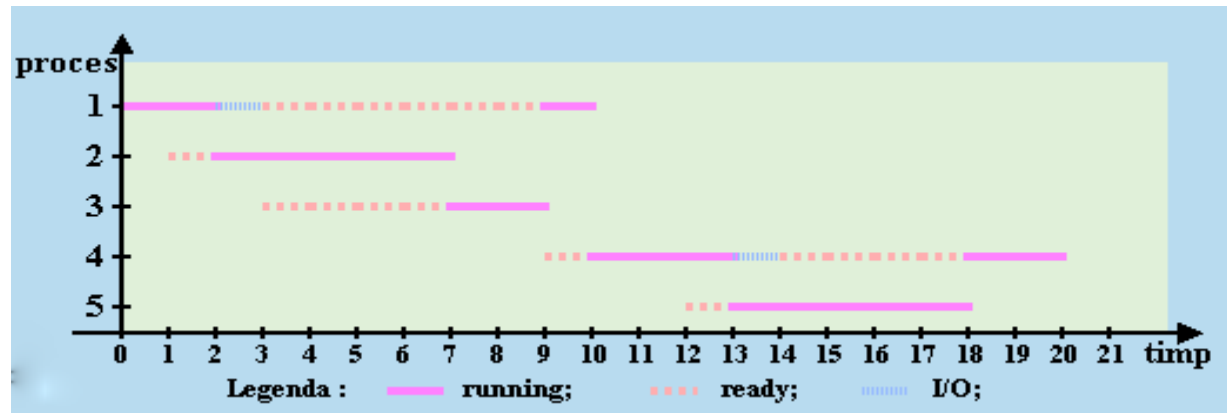


Algoritmi de planificare /8

• FCFS

Scenariul #2 (cu I/O)

Diagrama timp
a planificării
(pe 1 CPU) :



Rezultatele
planificării
FCFS:

Proces	Sosire	Serviciu	Start	Finish	T	W	P
A	0	2; 1 ;1	0	10	10	6	2.5
B	1	5	2	7	6	1	1.2
C	3	2	7	9	6	4	3.0
D	9	3; 1 ;2	10	20	11	5	1.83
E	12	5	13	18	6	1	1.2

Sumarul planificării:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
A	A	B	B	B	B	B	C	C	A	D	D	D	E	E	E	E	E	D	D	

Algoritmi de planificare /9

- **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 /10

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

- Planificarea SJF poate fi ne-preemptivă, sau preemptivă
- Varianta SJF preemptivă este numită planificare **shortest-remaining-time-first (SRTF)**
- SJF favorizează procesele interactive, ce necesită răspuns rapid și care interacționează cu utilizatorul în mod repetat
- SJF favorizează procesele ce produc rafale I/O (*I/O bursts*) – cele care se blochează curând, ținând perifericele ocupate și 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 /11

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

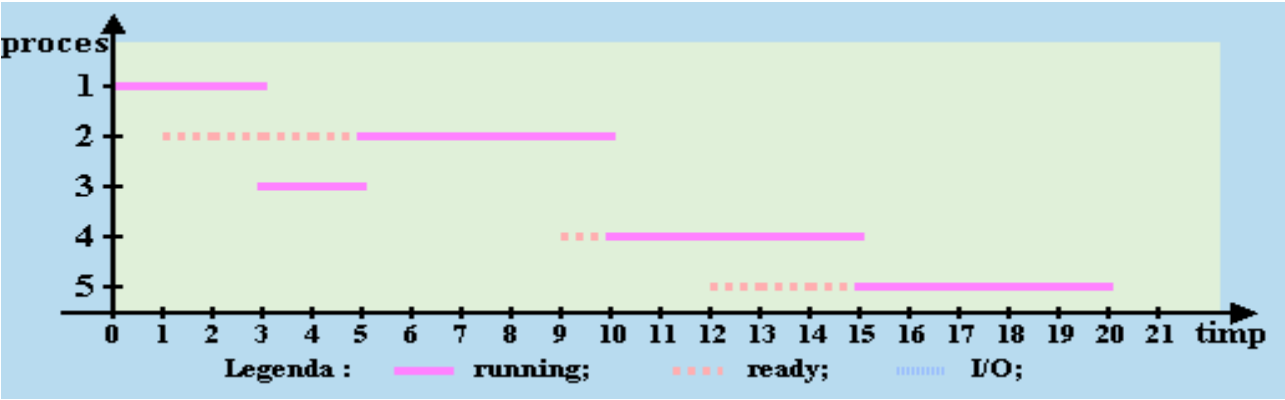
- Sacrifică echitatea pentru a micșora timpul mediu de răspuns
- Planificarea SJF pură este impracticabilă: planificatorul nu poate anticipa durata unui proces
- Totuși, SJF are valoare în sistemele reale:
 - Multe aplicații execută o secvență de rafale CPU scurte cu operații I/O între acestea
 - E.g., 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 /12

- **SJF**

Scenariul #1
(fără I/O)

Diagrama timp
a planificării
(pe 1 CPU) :



Rezultatele
planificării
SJF (i.e., alg.
ne-preemptiv):

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

Sumarul planificării:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
A	A	A	C	C	B	B	B	B	B	D	D	D	D	D	E	E	E	E	E	E

Algoritmi de planificare /13

- **SJF**

Temă (pentru acasă): aplicați SJF pentru **Scenariul #2** (cu I/O). Rezultatul planificării: similar ca la FCFS.

- **SRTF**

Scenariul #3 (alt exemplu, tot fără I/O)

Diagrama timp a planificării (pe 1 CPU) : ... (temă pentru acasă)

Rezultatele
planificării
SRTF (i.e.,
alg. preemptiv):

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	5	2	5	7	2	0	1.0
D	9	7	10	20	11	4	1.57
E	12	3	12	15	3	0	1.0

Sumarul planificării:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
A	A	A	B	B	C	C	B	B	B	D	D	E	E	E	D	D	D	D	D	

Algoritmi de planificare /14

- **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 priorități mari, sau invers)
- Observație: SJF poate fi privit ca un algoritm cu priorități, unde prioritatea (p) este inversul duratei următoarei rafale CPU anticipate (c) : $p = 1/c$

Algoritmi de planificare /15

- **Planificarea cu priorități**

- Prioritățile pot fi statice, sau dinamice
- Planificarea poate fi preemptivă, sau ne-preemptivă
- Problemă: *Blocarea nelimitată* sau *înfometarea* (*starvation*) proceselor cu priorități mici
- Soluție: *îmbătrânirea* (*aging*), i.e. creșterea graduală a priorității proceselor care așteaptă în coada ready pentru o perioadă mare de timp

- **Variantă: planificarea cu priorități cu time-slicing**

- Pentru procese ready cu priorități egale se aplică RR

Algoritmi de planificare /16

• Planificarea cu priorități

Scenariul #1' (fără I/O) – reconsiderăm primul exemplu, la care adăugăm priorități:

Diagrama timp a planificării (pe 1 CPU) : ... (*temă pentru acasă*)

Rezultatele
planificării
cu priorități
statice,
preemptivă:

Proces	Sosire	Serviciu	Prioritate	Start	Finish	T	W	P
A	0	3	1	0	20	20	17	6.6
B	1	5	2	1	8	7	2	1.4
C	3	2	4	3	5	2	0	1.0
D	9	5	3	9	19	10	5	2.0
E	12	5	5	12	17	5	0	1.0

Sumarul planificării:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
A	B	B	C	C	B	B	B	A	D	D	D	E	E	E	E	E	D	D	A	

• 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 de ordinul $10 \div 100$ ms
- 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 /18

• Planificarea Round-Robin (RR)

– *Timpul de viață*

RR reduce timpul de viață pentru procesele scurte

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

– *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.

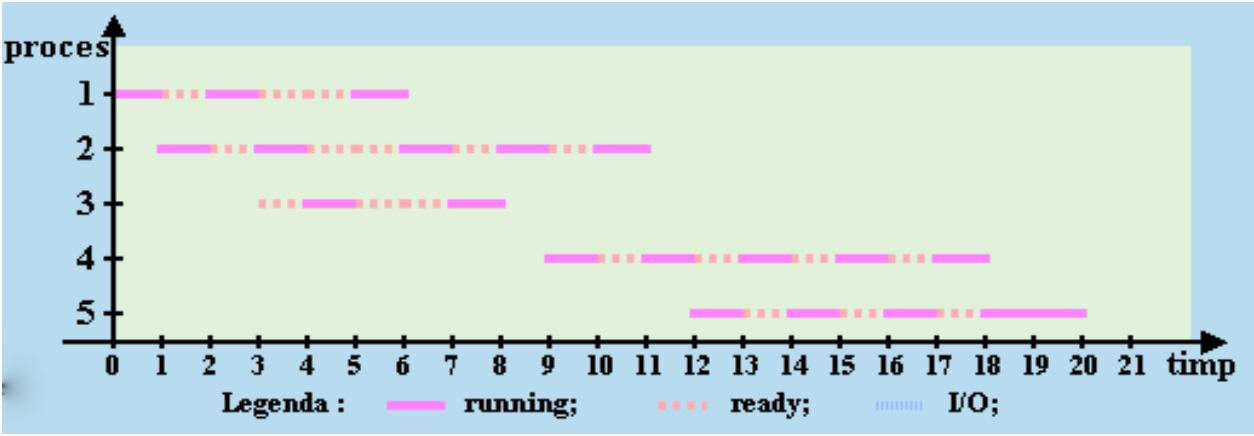
Pe un sistem *multiprocesor*, RR poate îmbunătăți rata de servire sub o încărcare ușoară (i.e. număr mic de joburi).

Algoritmi de planificare /19

- **RR**, cu cuanta = 1 unitate de timp

Scenariul #1 (fără I/O)

Diagrama timp
a planificării
(pe 1 CPU) :



Rezultatele
planificării RR,
cu cuanta=1:

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

Sumarul planificării:

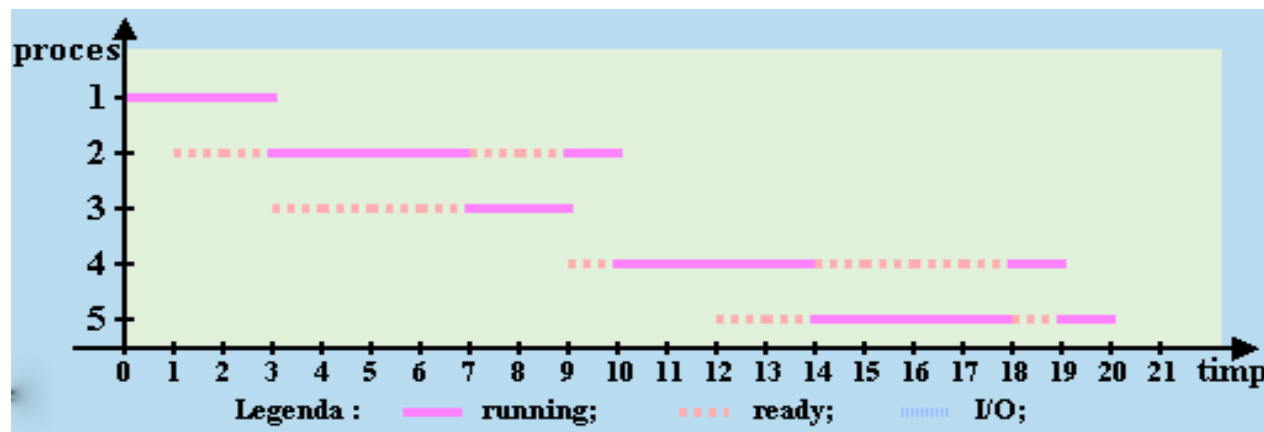
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
A	B	A	B	C	A	B	C	B	D	B	D	E	D	E	D	E	D	E	E	

Algoritmi de planificare /20

- **RR**, cu cuanta = 4 unități de timp

Scenariul #1 (fără I/O)

Diagrama timp
a planificării
(pe 1 CPU) :



Rezultatele
planificării RR,
cu cuanta=4:

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

Sumarul planificării:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
A	A	A	B	B	B	B	C	C	B	D	D	D	D	E	E	E	E	D	E	

Algoritmi de planificare /21

• Planificarea Round-Robin (RR)

- Problemă: Cât ar trebui să fie cuanta de timp ?
- Dacă e prea mare, pur și simplu RR devine FIFO (i.e., FCFS)
- 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)

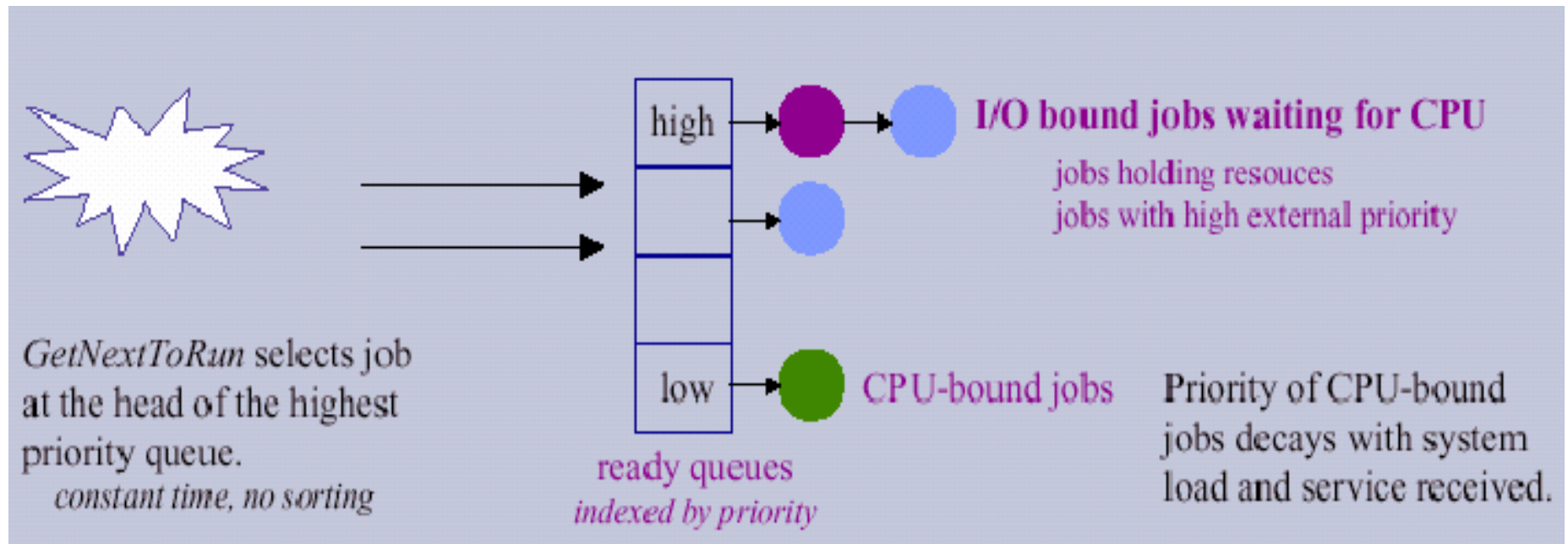
Algoritmi de planificare /22

- **Planificarea cu cozi pe nivele multiple**

- Divizarea cozii ready în mai multe nivele distincte, i.e. mai multe (sub-)cozi ready separate
E.g., coada proceselor *foreground* (joburi interactive) și coada proceselor *background* (batch-joburi)
- 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 și o “planificare între cozi” (e.g., poate fi utilizată o planificare preemptivă cu priorități fixe)

Algoritmi de planificare /23

- **Planificarea cu cozi pe nivele multiple**
 - Exemplu:

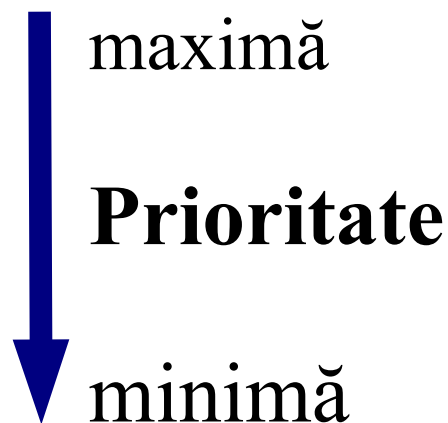


Algoritmi de planificare /24

- **Planificarea cu cozi pe nivele multiple**

- Alt exemplu:

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



Algoritmi de “planificare între cozi” (i.e., de alegere a uneia dintre cozi):

- O soluție, pe baza priorității cozilor: fiecare coadă are întâietate absolută asupra cozilor cu priorități mai joase
- Altă soluție, RR: de împărțit timpul în cuante (*time slice*) între cozi

Algoritmi de planificare /25

- **Planificarea cu cozi pe nivele multiple, cu feedback**
 - Se permite unui proces să migreze între cozi
 - Parametri:
 - Numărul de cozi + algoritmul de “planificare între cozi”
 - Algoritmii de planificare pentru fiecare coadă
 - Metoda utilizată pentru a decide în ce coadă va intra inițial un proces (i.e., atunci când acesta este lansat în execuție)
 - 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
 - Exemplu: Windows utilizează o coadă ready pe 32 nivele de prioritate

Algoritmi de planificare /26

- **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 /27

- **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 (multiprocesare simetrică): sisteme omogene, în care procesele sunt planificate pe oricare dintre procesoarele disponibile
 - Planificare master-slave (multiprocesare asimetrică)
 - *Processor affinity*: afinitate hard (un proces poate specifica să fie rulat doar pe un subset din CPU-urile existente) vs. afinitate soft (SO-ul încearcă să păstreze un proces pe același CPU, pentru a reaccesa informațiile din cache-ul CPU-ului, sau în cazul sistemelor NUMA, pentru acces local)

Algoritmi de planificare /25

• Planificarea multiprocesor

Scenariul #4 (alt exemplu, tot fără I/O): planificare SMP (cu două CPU),
cu o singură coadă ready

Diagrama timp a planificării (pe 2 CPU) : ... (*temă pentru acasă*)

Rezultatele
planificării
cu priorități
statice,
preemptivă,
pe 2 CPU:

Proces	Sosire	Serviciu	Prioritate	Start	Finish	T	W	P
A	0	3	1	0	5	5	2	1.66
B	1	7	2	1	12	11	4	1.57
C	2	2	4	2	4	2	0	1.0
D	6	5	3	6	11	5	0	1.0
E	7	5	5	7	12	5	0	1.0

Sumarul
planificării:

Procesor	00	01	02	03	04	05	06	07	08	09	10	11	12	13
CPU1	A	A	C	C	A		D	D	D	D	D	B		
CPU2		B	B	B	B	B	B	E	E	E	E	E		

- **Bibliografie obligatorie**
capitolele despre *gestiunea proceselor* din
 - Silberschatz : “*Operating System Concepts*”
(cap.5 din [OSC10])
 - sau
 - Tanenbaum : “*Modern Operating Systems*”
(a patra parte a cap.2 din [MOS4])

- Planificarea proceselor (continuare)
 - Structura planificării
 - Algoritmi de planificare:
 - FCFS
 - SJF și SRTF
 - Planificarea cu priorități
 - RR
 - ș.a.

Întrebări ?