

Sisteme de Operare

Gestiunea proceselor – partea I

Cristian Vidrașcu

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

Cuprins

- Conceptul de proces
- Stările procesului
- Relații între procese
- Procese concurente
- Planificarea proceselor

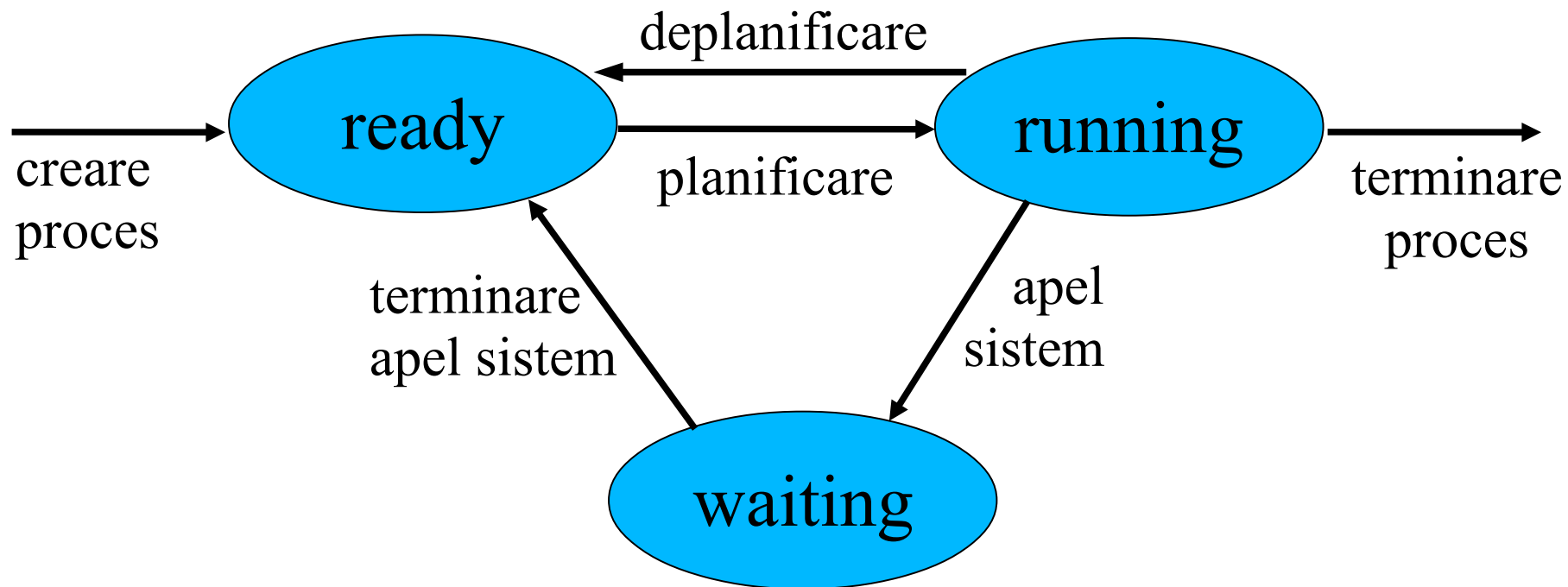
Conceptul de proces

- Proces (task, job)
 - **Job** : un program în curs de execuție, fiind o secvență formată din unul sau mai multe procese
 - **Proces** : o entitate activă (dinamică) a S.O.-ului, fiind unitatea de lucru într-un sistem de calcul
- Un proces include, printre alte resurse:
 - **zona de cod** (ce conține codul programului)
 - **zona de date** (ce conține variabilele globale)
 - **stiva de lucru** (ce conține informații temporare, e.g. parametrii subrutinelor, adresele de return, variabilele temporare)

Stările procesului /1

- Pe parcursul execuției sale, un proces își schimbă **starea**
- Fiecare proces (mai exact, fir de execuție) poate fi într-una din următoarele stări:
 - **running** (în execuție)
 - **waiting** (în așteptare)
 - **ready** (gata de execuție)
- În orice moment, un singur proces poate fi în starea running (în cazul sistemelor uniprocessor)

Stările procesului /2



Blocul de control al procesului

- **PCB** (Process Control Block) este o structură de date reprezentând un proces în cadrul S.O.-ului, ce păstrează următoarele informații:
 - ID-ul procesului (i.e. PID-ul)
 - PID-ul procesului părinte (cel care a creat respectivul proces)
 - starea procesului
 - contorul de program (*program counter*) și ceilalți regiștri CPU
 - directorul curent de lucru; linia de comandă; variabilele de mediu
 - drepturile de acces la resursele sistemului
 - fișierele deschise de respectivul proces
 - informații de planificare a CPU
 - informații de gestiune a memoriei
 - informații pentru raportări (*accounting*)
 - informații despre starea I/E

ș.a.

Relații între procese /1

- Un proces este **independent** dacă nu poate afecta și nici nu poate fi afectat de celelalte procese ce se execută în sistem
 - “starea” procesului **nu este partajată** de alte procese
 - execuția procesului este **deterministă** (depinde în întregime numai de datele de intrare)
 - execuția procesului este **reproductibilă** (rezultatul execuției va fi mereu același pentru aceleași date de intrare)
 - execuția procesului **poate fi suspendată** și apoi **poate fi reluată** fără a cauza efecte nedorite

Relații între procese /2

- Un proces este **cooperant** dacă poate afecta sau poate fi afectat de celelalte procese ce se execută în sistem
 - “starea” procesului **este partajată** de alte procese
 - rezultatul execuției procesului **nu poate fi prevăzut** în avans
 - rezultatul execuției procesului este **nedeterminist** (nu depinde numai de datele de intrare)

Relații între procese /3

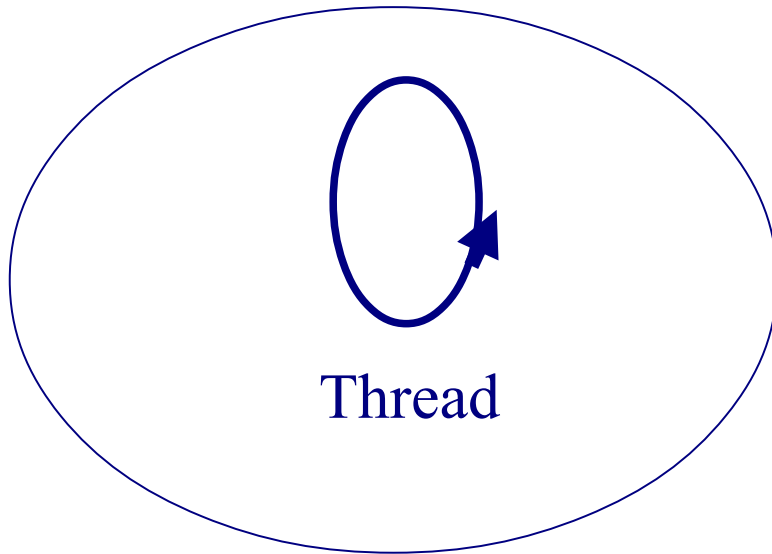
- De ce se utilizează mai multe procese / thread-uri ?
 - Pentru a captura activități natural concurente în cadrul sistemului programat
 - Tratarea evenimentelor asincrone
 - Pentru a câștiga viteză de execuție (*speedup*) prin suprapunerea activităților de calcul cu cele de I/E sau prin exploatarea hardware-ului paralel

Relații între procese /4

- Abstractizarea **thread** – definește un singur flux secvențial de instrucțiuni (contor program, stivă, valori regiștri)
 - Un thread este unitatea de bază de utilizare a CPU
 - Poate fi suportat de nucleul S.O.-ului (e.g. OS/2, Windows NT/2000/XP/2003/Vista/2008/7/8..., UNIX)
- **Proces** – resursa context, cu rol de “container” pentru unul sau mai multe thread-uri (spațiu de adrese partajat de către acestea)

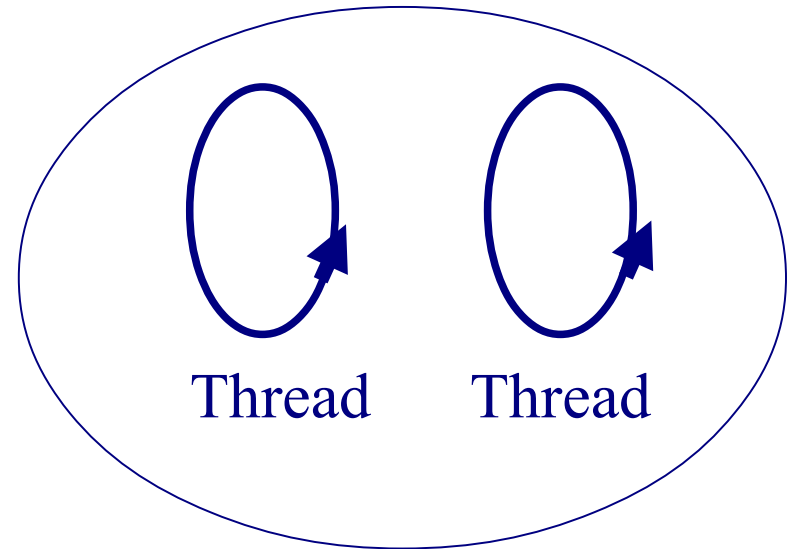
Relații între procese /5

Thread



Address Space

Process



Address Space

Procese concurente /1

- Procese **multiple** pot fi multiprogramate pe un **singur** CPU
- Motive:
 - partajarea resurselor fizice
 - partajarea resurselor logice
 - creșterea vitezei de calcul (*speedup* computațional)
 - modularitate
 - comoditate de utilizare a sistemului

Procese concurente /2

- Crearea și terminarea proceselor (modelul Unix)

```
int pid;  
int status = 0;  
  
if (pid = fork())  
{ /* parent */  
    .....  
    pid = wait (&status);  
}  
else  
{ /* child */  
    .....  
    exit(status);  
}
```

*Primitiva **fork** returnează zero fiului și PID-ul fiului părintelui*

***Fork** creează o copie exactă a procesului părinte*

*Părintele folosește **wait** pentru a dormi până când fiul se termină; apelul wait returnează PID-ul fiului și codul de terminare. Variantele de **wait** permit așteptarea unui anumit fiu, sau notificarea stopării fiului sau a altor semnale.*

*Fiul întoarce părintelui codul de terminare la **exit**, pentru a raporta succesul/eșecul.*

- Procese **părinte** și **fiu**

Procese concurente /3

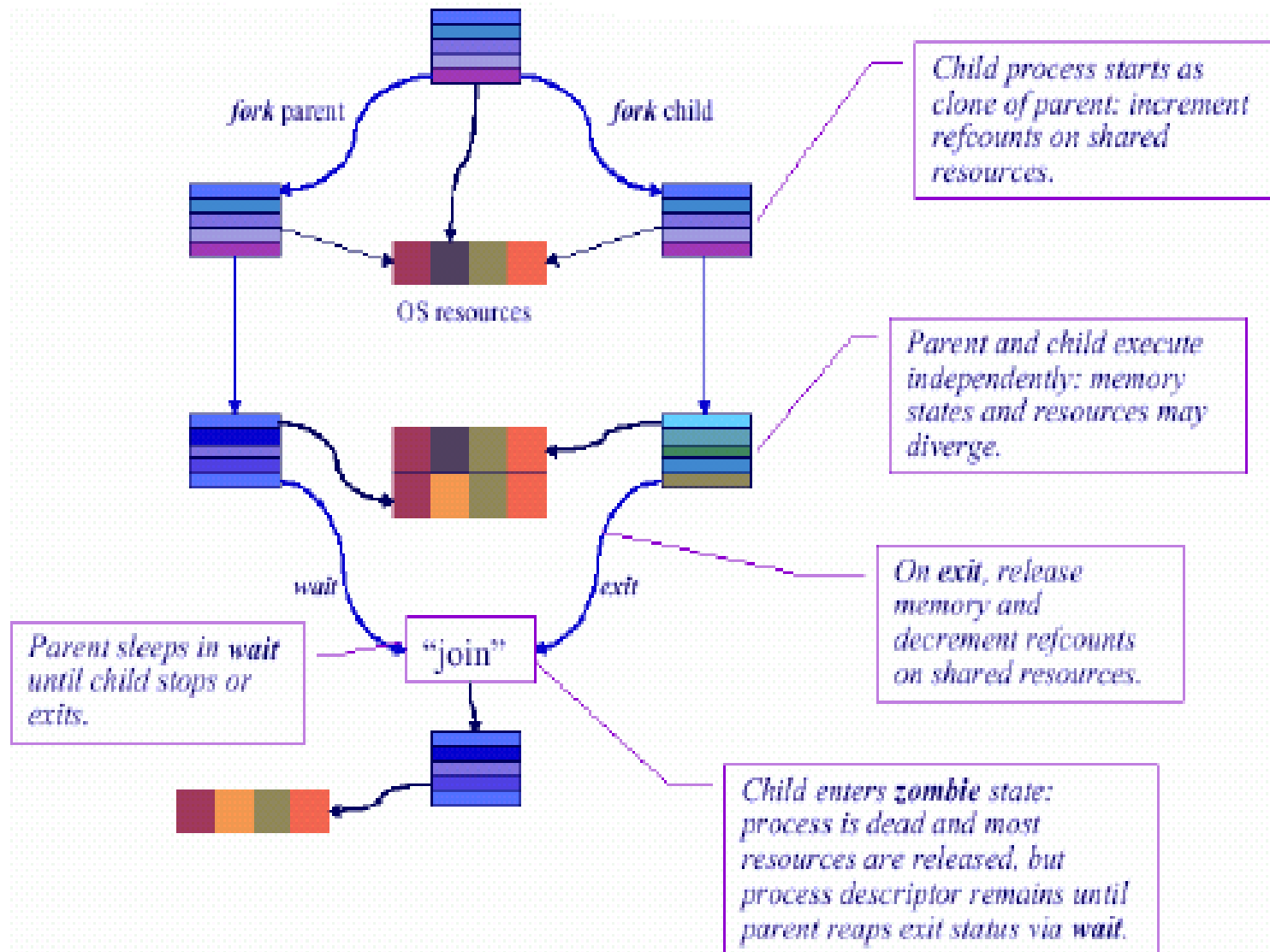
- Disciplina fiului
 - După un apel `fork()`, programul părinte are controlul total asupra comportamentului fiului său
 - Fiul își moștenește mediul de execuție de la părinte (dar programul părinte îl poate schimba)
 - asignările descriptorilor de fișiere sunt setate cu `open()`, `close()`, `dup()`
 - `pipe()` inițializează canalele de comunicație între procese
 - Programul părinte poate pune fiul să execute un program diferit, apelând `exec()` în contextul fiului

Procese concurente /4

- Procesul fiu trebuie să poată fi diferit de părinte
 - Primitivele `exec()` “bootează” fiul cu o imagine executabilă diferită de cea a părintelui
 - programul părinte apelează primitiva `exec()` (în contextul fiului creat) pentru a executa în acesta un nou program
 - `exec()` reacoperă procesul fiu cu o nouă imagine executabilă
 - restartează fiul în mod utilizator la un punct de intrare predeterminat
 - nu returnează nici o valoare programului apelant (acesta nu mai există)
 - argumentele liniei de comandă și variabilele de mediu sunt transferate în memorie
 - descriptorii de fișiere, PID-ul, ș.a. rămân neschimbate

Procese concurente /5

Exemplu



Procese concurente /6

- Mai multe cazuri trebuie considerate pentru “join”
e.g. `exit()` și `wait()`
 - Ce se întâmplă dacă fiul face `exit` (se termină) înainte ca tatăl să facă `join`?
 - Un obiect proces “zombie” păstrează codul de terminare și informațiile de stare ale fiului
 - Ce se întâmplă dacă tatăl se termină înaintea fiului?
 - Orfanii devin copii ai procesului **init** (cu PID-ul 1)
 - Ce se întâmplă dacă tatăl nu-și poate permite să aștepte la un punct de `join`?
 - Facilități pentru notificări asincrone (prin semnale Unix)

Planificarea proceselor

- Planificarea proceselor (va fi continuată)
 - Obiective
 - Cozi de planificare
 - Planificatoare
 - Structura planificării
 - Schimbarea contextului
 - Priorități
 - Algoritmi de planificare

Planificarea proceselor /1

- Obiectivele gestiunii procesorului
 - de a aloca timp CPU la joburile/procese de executat, într-o asemenea manieră încât să optimizeze un anumit aspect (sau mai multe aspecte) ale performanței utilizării sistemului de calcul

Planificarea proceselor /2

Obiective urmărite:

- **Echitate**

Asigurarea faptului că fiecare proces are șanse echitabile la CPU

- **Timp de răspuns**

Minimizarea timpului de răspuns pentru utilizatorii interactivi

- **Predictibilitate**

Asigurarea faptului că un același job va avea o aceeași durată de execuție indiferent de variabilele sistemului

- **Eficiența**

Furnizarea unui grad ridicat de utilizare a CPU

- **Utilizarea resurselor**

Asigurarea faptului că toate resursele sunt folosite la maxim

- **Throughput** (rata de servire)

Maximizarea numărului de joburi executate pe oră

- **Evitarea amânării la infinit**

Asigurarea faptului că toate joburile se termină de executat

- **Deadlines** (termene limită)

Asigurarea îndeplinirii termenelor limită specificate de utilizatori

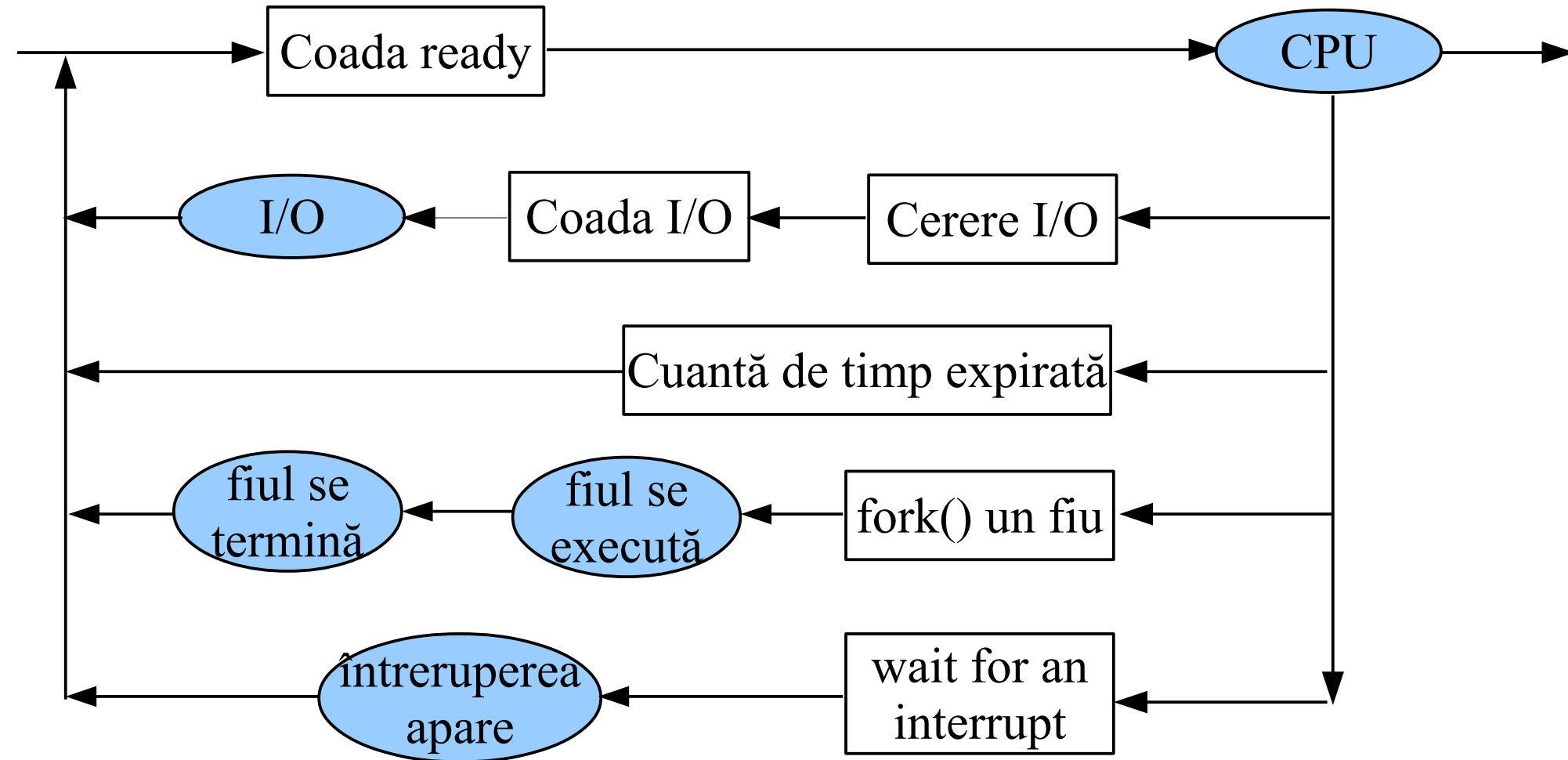
Cozi de planificare /1

- Cozi de planificare
 - pe măsură ce procesele intră în sistem, sunt depuse într-o **coadă de joburi** (cu toate procesele ce așteaptă să li se aloce memoria principală)
 - procesele ce sunt rezidente în memoria principală și care sunt gata de execuție și așteaptă să fie executate, sunt păstrate în **coada ready** (gata de execuție)
 - procesele ce așteaptă un dispozitiv periferic I/O sunt păstrate într-o **coadă I/O** (coada periferic)

Cozi de planificare /2

- Un nou proces este pus inițial în coada ready
- El așteaptă în coada ready până când este selectat pentru execuție și i se dă CPU-ul
- După ce CPU-ul îi este alocat procesului și începe să-l execute, pot apare mai multe evenimente:
 - procesul poate lansa o cerere I/O și apoi este plasat într-o coadă I/O
 - procesul poate `fork()` un nou proces și `wait()` terminarea fiului
 - procesul poate fi înlăturat forțat de pe CPU (ca urmare a unei întreruperi) și plasat înapoi în coada ready

Cozi de planificare /3



Planificatoare /1

Activitatea S.O.-ului de planificare poate fi considerată că se desfășoară la trei nivele:

- 1. La nivelul înalt (planificarea joburilor)** se decide care joburi pot intra în sistem pentru a concura pentru resursele acestuia
- 2. La nivelul de mijloc (planificarea proceselor)** se ajustează prioritățile proceselor și se pot suspenda procese, determinând astfel care procese vor concura pentru CPU
- 3. La nivelul scăzut (dispecerat)** se decide cărui thread i se va da efectiv CPU-ul

Planificatoare:

1. Planificator pe termen lung (planificator de joburi)

- selectează procesele și le încarcă în memorie pentru execuție
- controlează *gradul de multi-programare* (i.e., numărul de procese din memorie)

2. Planificator pe termen scurt (planificator CPU)

- selectează dintre procesele care sunt în starea ready (gata de execuție) unul căruia îi alocă CPU-ul pentru următoarea cantă de timp procesor
- trebuie să fie foarte rapid, deoarece va fi executat cel puțin o dată la ~ 10 ms

Tipuri de procese:

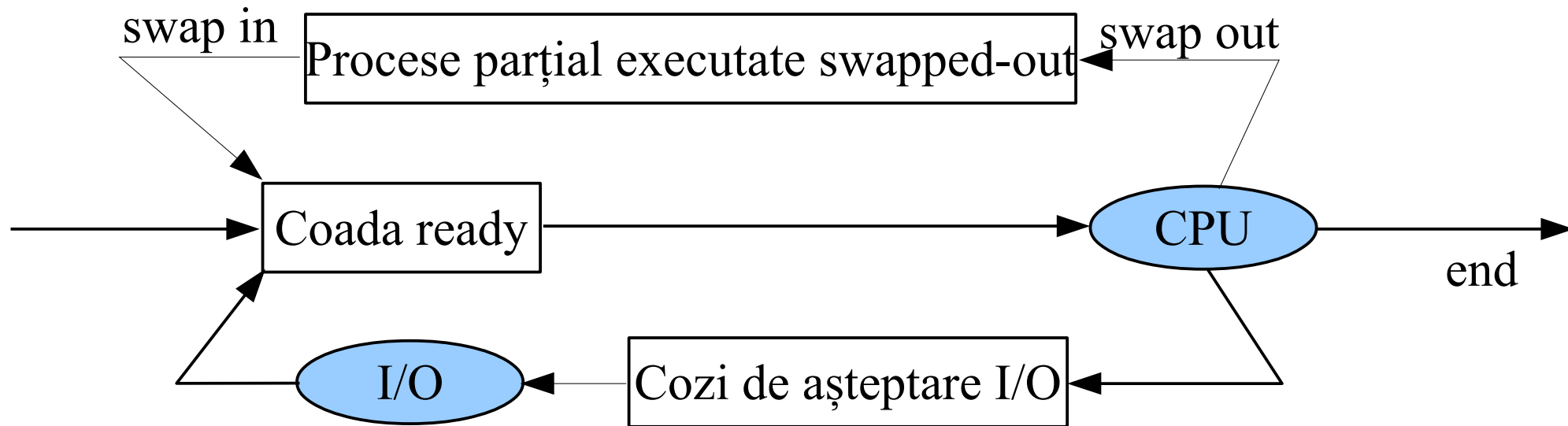
- Procese I/O-intensive
 - un proces care generează des cereri I/O, i.e. care-și petrece mai mult timp făcând operații I/O decât efectuând calcule
- Procese CPU-intensive
 - un proces care generează rar cereri I/O, petrecându-și timpul mai mult făcând calcule decât operații I/O

Planificatoare /4

- Orice algoritm de planificare trebuie să ia în calcul următorii factori:
 - i) dacă un task este I/O-intensiv sau CPU-intensiv,
 - ii) dacă un task este de tip batch sau interactiv, și
 - iii) cât de urgent se cere a fi răspunsul.
- Sistemul cu cea mai bună performanță va avea o combinație de procese CPU-intensive și I/O-intensive.
- Pentru sistemele moderne, planificatorul pe termen lung poate fi minimal sau chiar absent.

Planificatoare /5

- Unele S.O.-uri (e.g. sisteme cu time-sharing) pot introduce un nivel intermediar de planificare:
planificatorul pe termen mediu



Schema de swapping

- **Bibliografie obligatorie**
capitolele despre *gestiunea proceselor* din
 - Silberschatz : “*Operating System Concepts*”
(cap.3,5 din [OSCE8])
 - sau
 - Tanenbaum : “*Modern Operating Systems*”
(prima parte a cap.2 din [MOS3])

Sumar

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

va fi continuată

Întrebări ?