Sisteme de Operare

Gestiunea proceselor
partea I-a

Cristian Vidrașcu

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

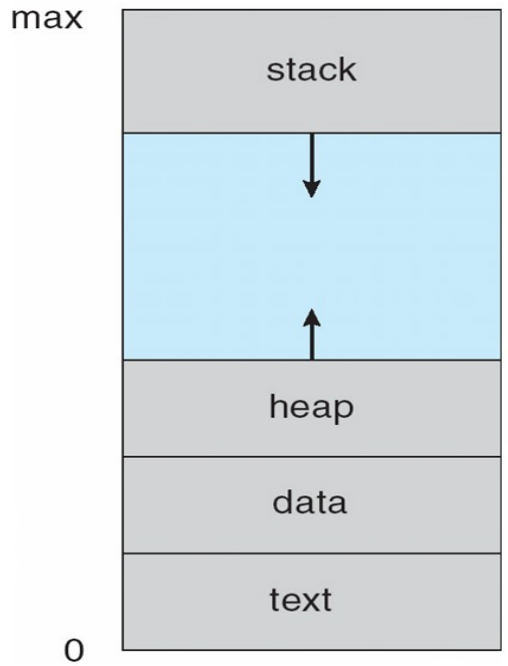
Cuprins

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

Conceptul de proces

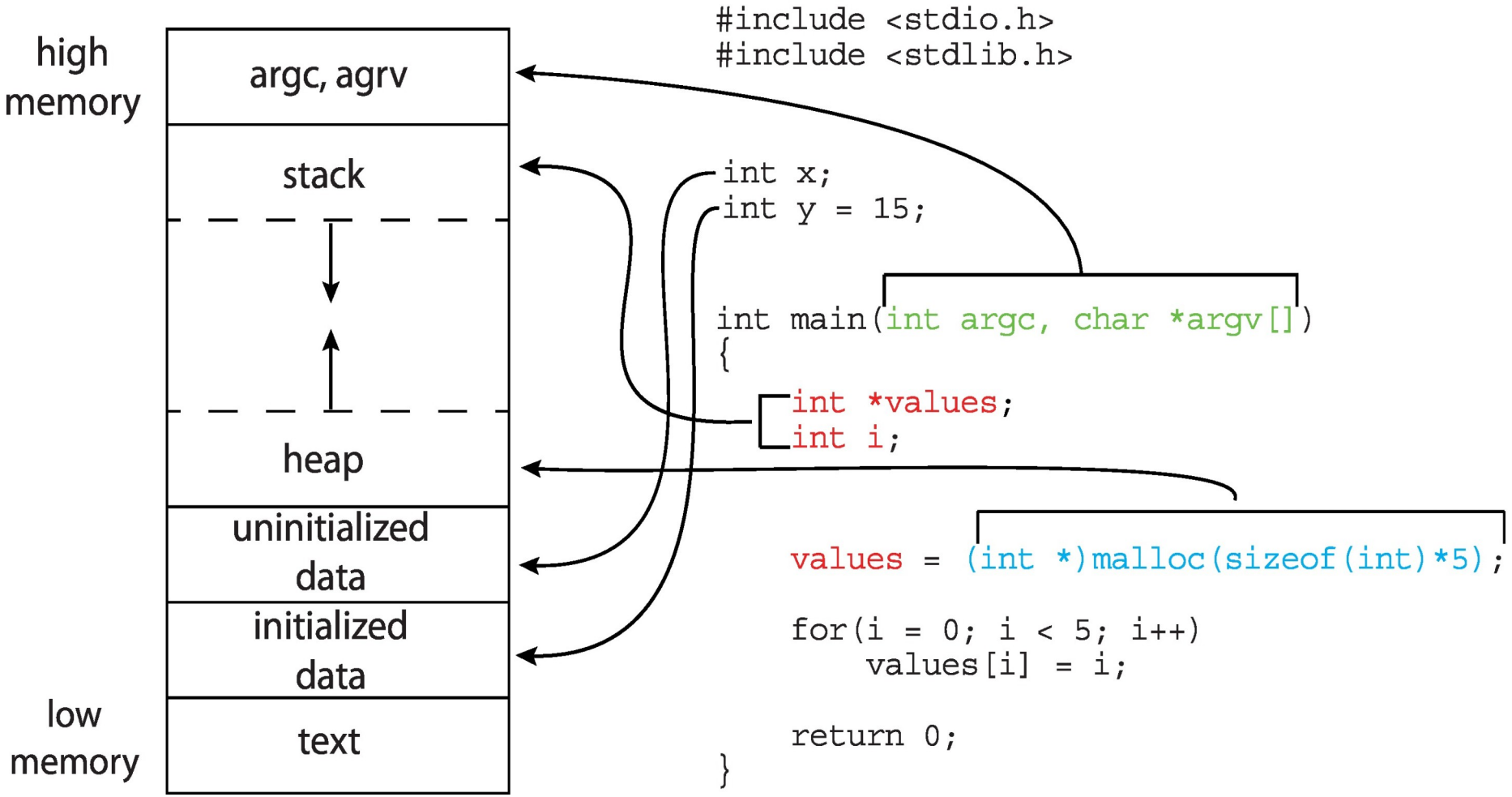
- Proces (sau task) vs job
  - **Job** : un program în curs de execuție, fiind o secvență formată din unul sau mai multe procese (e.g. Google Chrome)
  - **Proces** : o entitate activă (dinamică) a S.O.-ului, fiind unitatea de lucru tradițională într-un sistem de calcul

- Un proces include, printre alte resurse:
  - zona de cod (ce conține codul programului)
  - zona de date (ce conține variabilele globale)
  - zona de *heap* (pentru variabile alocate dinamic)
  - stiva de lucru (ce conține informații temporare, e.g. parametrii subrutinelor, adresele de return, variabilele temporare, etc.)



Conceptul de proces

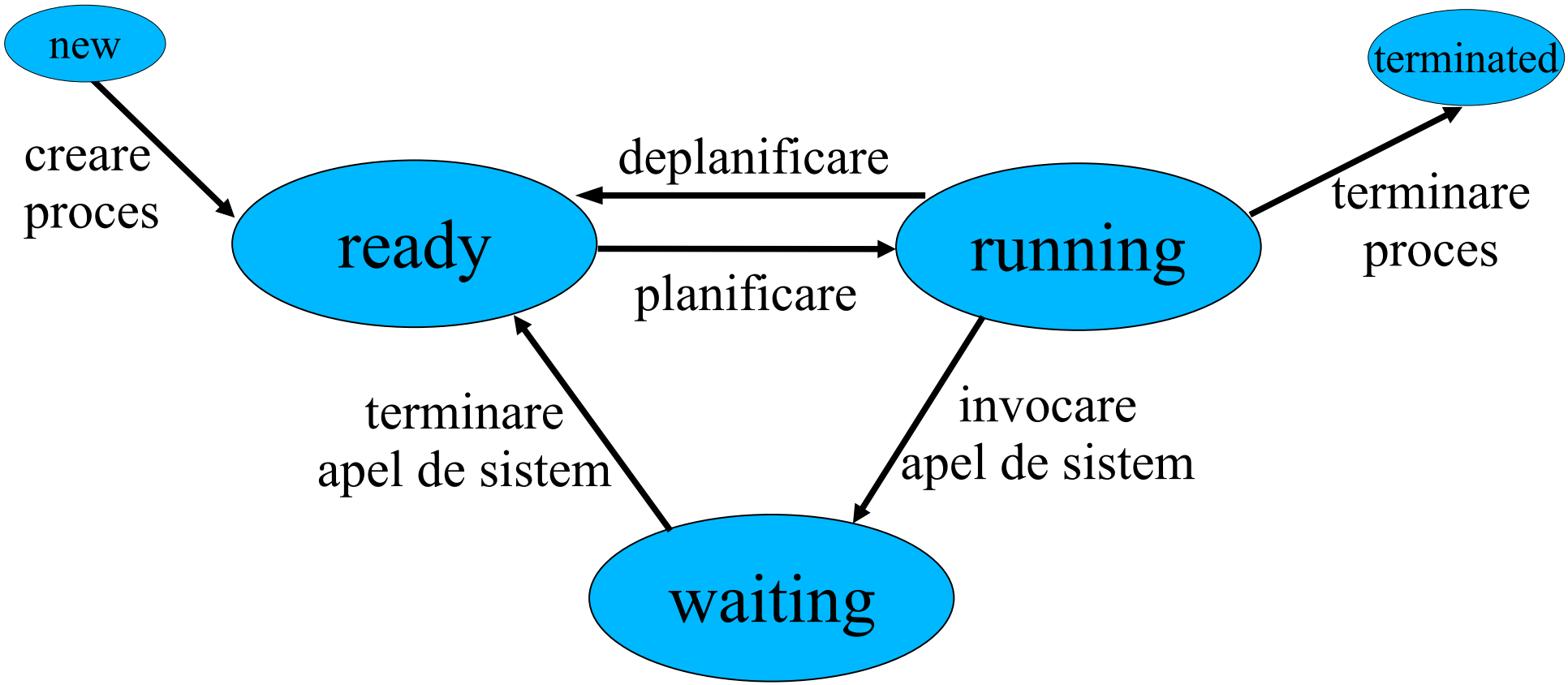
- *Layout*-ul memoriei unui program C:



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 cazul sistemelor uniprocesor, în orice moment, un singur proces (fir de execuție) poate fi în starea running!

Stările procesului /2



Notă: în sistemele bazate pe *multi-threading*, diagrama stărilor este la nivel de thread.

Blocul de control al procesului/1

- **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.

Blocul de control al procesului/2

- Ca și implementare, tipul de date PCB este un struct având câte un câmp pentru fiecare dintre informațiile enumerate pe slide-ul anterior, cu observația că unele câmpuri sunt la rândul lor mai complexe, fiind fie struct-uri, fie chiar pointeri către alte tipuri de date complexe

- **Tabela proceselor** = ansamblul tuturor structurilor PCB pentru toate procesele existente la un moment dat

- Ca și implementare, tabela proceselor poate fi:
  - un vector de PCB-uri, indexat după PID
  - o listă (simplu, sau dublu) înlănțuită de PCB-uri, eventual sortată după PID
  - 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)

Notă: se formează astfel grupuri de câte două sau mai multe procese cooperante, ce colaborează între ele pentru a-și îndeplini sarcinile.

Relații între procese /3

- De ce se utilizează mai multe procese cooperante (în cadrul unui job) ? Sau, de ce se utilizează mai multe fire de execuție (în cadrul unui proces) ?

– 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

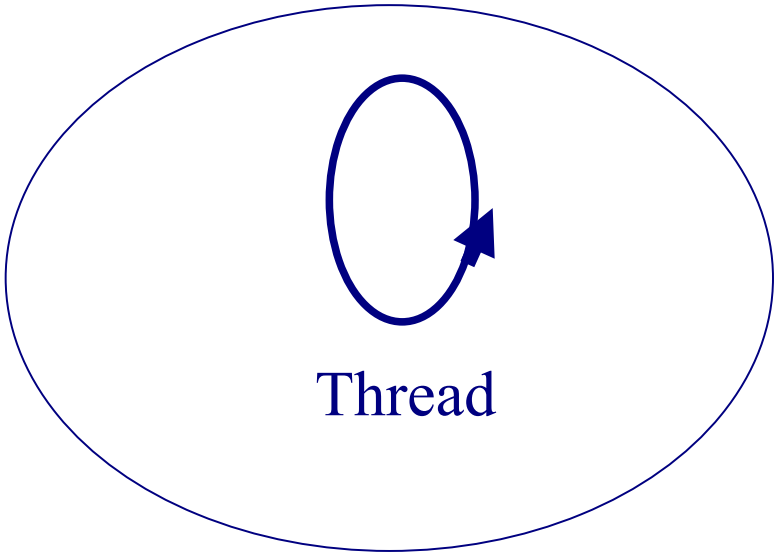
Relații între procese /4

- 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

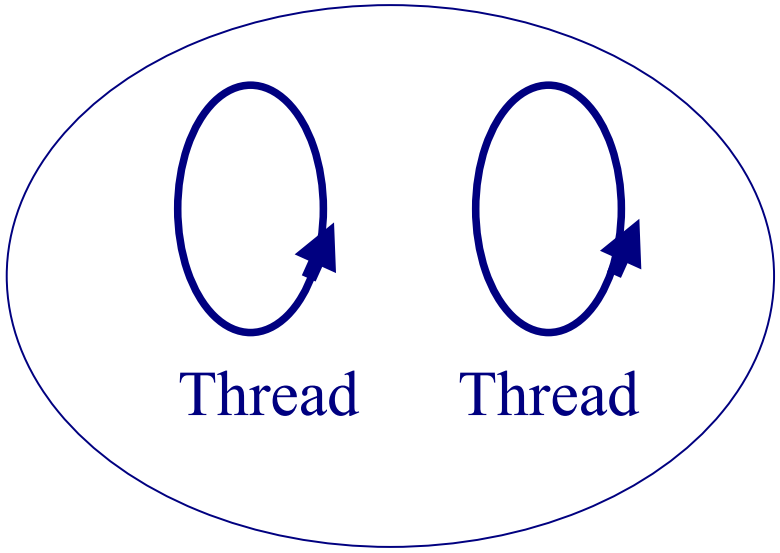
    e.g. OS/2, Windows NT (NT4/2000/.../Win8/Win10), Solaris și alte variante de UNIX
- **Proces** – resursa context, cu rol de "container" pentru unul sau mai multe thread-uri (spațiu de adrese partajat de către acestea)

Relații între procese /5

Proces secvențial
(cu un singur thread)



Proces multithreaded
(cu mai multe thread-uri)



Procese concurente /1

- Procese multiple pot fi multiprogramate pe un singur CPU (i.e., executate prin "paralelism aparent")
- 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

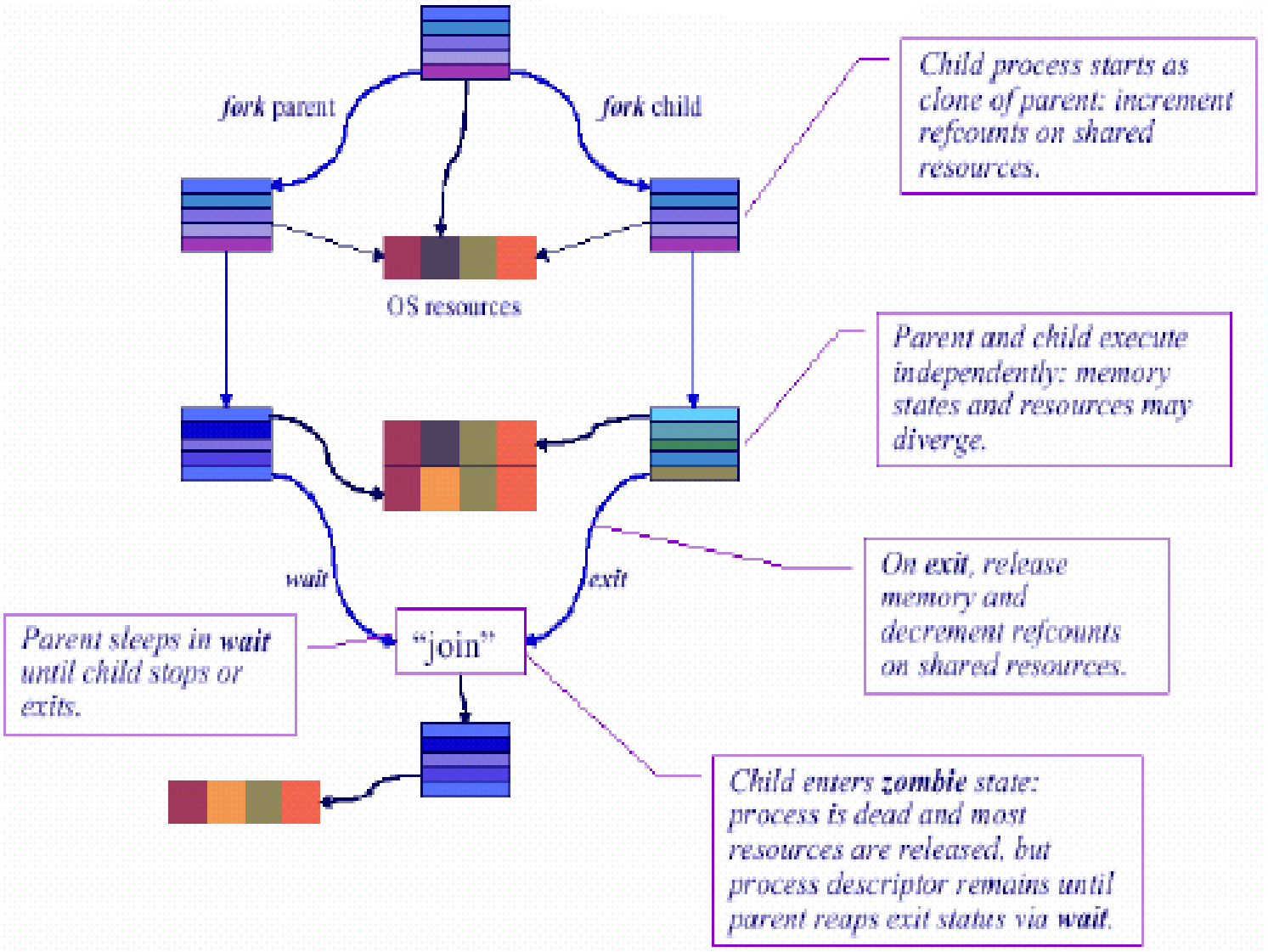
- Disciplina fiului
  - – După un apel fork(), programul părinte are controlul total asupra comportamentului fiului său
  - – Fiul își moștenește mediul de execuție de la părinte (dar programul părinte îl poate schimba)
    - asignările descriptorilor de fișiere sunt setate cu open(), close(), dup()
    - 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 (deoarece acesta nu mai există!)
    - argumentele liniei de comandă și variabilele de mediu sunt transferate în memorie → Declarația completă a funcției main: …
    - descriptorii de fișiere, PID-ul, ş.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
  - Schimbarea contextului
  - Priorități
  - Structura planificării
  - Algoritmi de planificare

Planificarea proceselor /1

- Obiectivele gestiunii procesorului
  - de a aloca timp CPU la joburile/procesele 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

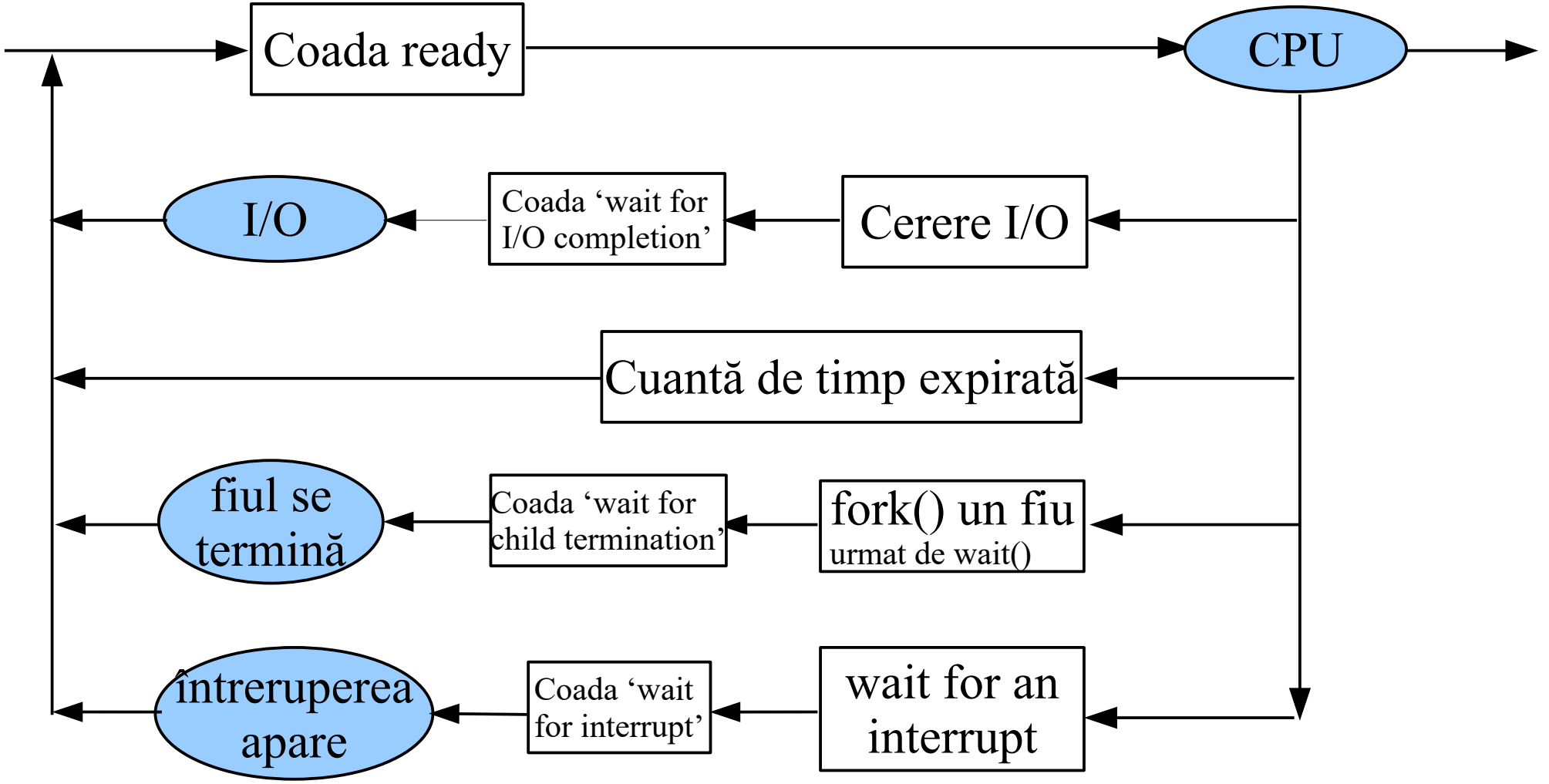
– pe măsură ce procesele întră în sistem, sunt depuse într-o **coadă de joburi** (cu toate procesele ce așteaptă să li se aloce memoria principală)
– procesele ce sunt rezidente în memoria principală și care sunt gata de execuție și așteaptă să fie executate, sunt păstrate în **coada ready** (gata de 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

Activitatea de planificare:



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 scazut (dispecerat)** se decide cărui thread i se va aloca efectiv CPU-ul

Planificatoare /2

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 (mai exact, dintre thread-urile) care sunt în starea ready (i.e., gata de execuție), unul căruia îi alocă CPU-ul pentru următoarea cuantă de timp procesor
   - acest planificator trebuie să fie foarte rapid, deoarece va fi executat cel puțin o dată la (aprox.) 10 ms

Planificatoare /3

Tipuri de procese:

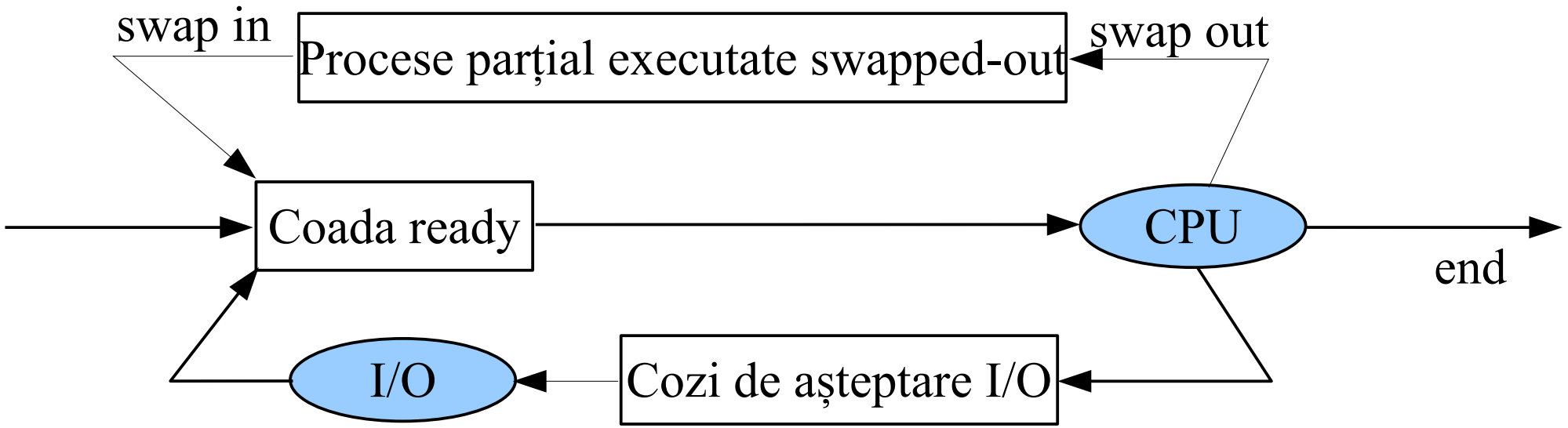
- Procese I/O-intensive
  - un proces care generează des cereri I/O, i.e. care-și petrece timpul mai mult făcând operații I/O decât efectuând calcule
- Procese CPU-intensive
  - un proces care generează rar cereri I/O, petrecându-și timpul mai mult făcând calcule decât operații I/O

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**

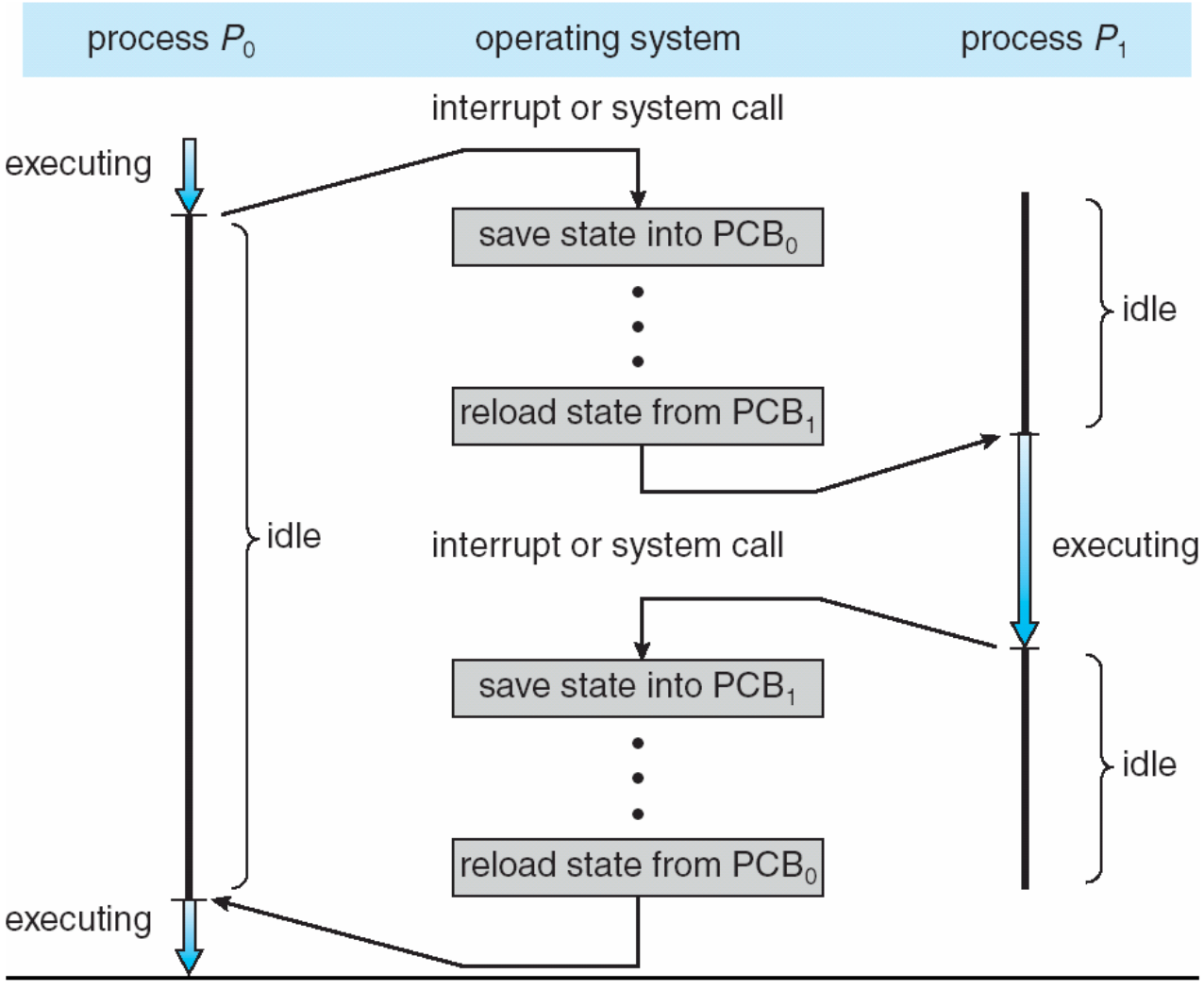


Schema de swapping

Schimbarea contextului/1

- *Context switch*

  Comutarea CPU-ului către alt proces necesită salvarea stării vechiului proces și încărcarea stării salvate anterior a noului proces ce urmează să se execute pe CPU.



Schimbarea contextului/2

– Timpul necesar pentru schimbarea contextului constituie o încărcare suplimentară a sistemului (de ordinul 1÷100 μs), dar depinde foarte mult de suportul oferit de hardware

– ***Dispatcher***: rutina din nucleul SO-ului care predă controlul CPU-ului către procesul selectat din coada ready de către *CPU scheduler* (i.e. planificatorul pe termen scurt), ceea ce presupune următoarele acțiuni:

1) Schimbarea contextului
2) Revenirea în modul neprivilegiat al procesorului (*user-mode*)
3) Salt la instrucțiunea potrivită din program (pe baza informațiilor salvate anterior în contextul acelui proces) pentru reluarea execuției acestuia

Notă: rutina *dispatcher* este scrisă adesea în limbaj mașină, pentru eficiență la execuție

Priorități /1

- **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

- Manipularea priorităților
  - intern – planificatorul calculează dinamic prioritățile și le utilizează pentru gestiunea cozilor de planificare

    (i.e., sistemul ajustează intern valorile priorităților, pe parcursul execuției joburilor, printr-o tehnică implementată în planificator)
  - extern – valori statice, în funcție de rangul utilizatorului, ș.a.

- 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

Priorități /3

Prioritățile trebuie manevrate cu grijă atunci când există dependințe 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.

Bibliografie

- **Bibliografie obligatorie**

  capitolele despre *gestiunea proceselor* din

  – Silberschatz : "*Operating System Concepts*"

  (cap.3,5 din [OSC10])

  sau

  – Tanenbaum : "*Modern Operating Systems*"

  (prima parte a cap.2 din [MOS4])

Sumar

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

va fi continuată

Întrebări ?