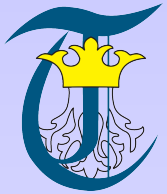


Universitatea *Transilvania* Brașov
Facultatea de Inginerie Electrică și Știința Calculatoarelor
Departamentul de Electronică și Calculatoare

Sisteme de operare

– curs 3 –

2013.03.13

ș.l. dr. ing. Kertész Csaba-Zoltán

2. Procese

Cuprins

2.1 Introducere

- un proces este un program aflat în execuție
- fiecărui proces îi sunt alocate niște resurse necesare pentru realizarea sarcinii
  - timp procesor
  - memorie
  - fișiere și dispozitive I/O
- un sistem este alcătuit din mai multe procese ce lucrează concurent

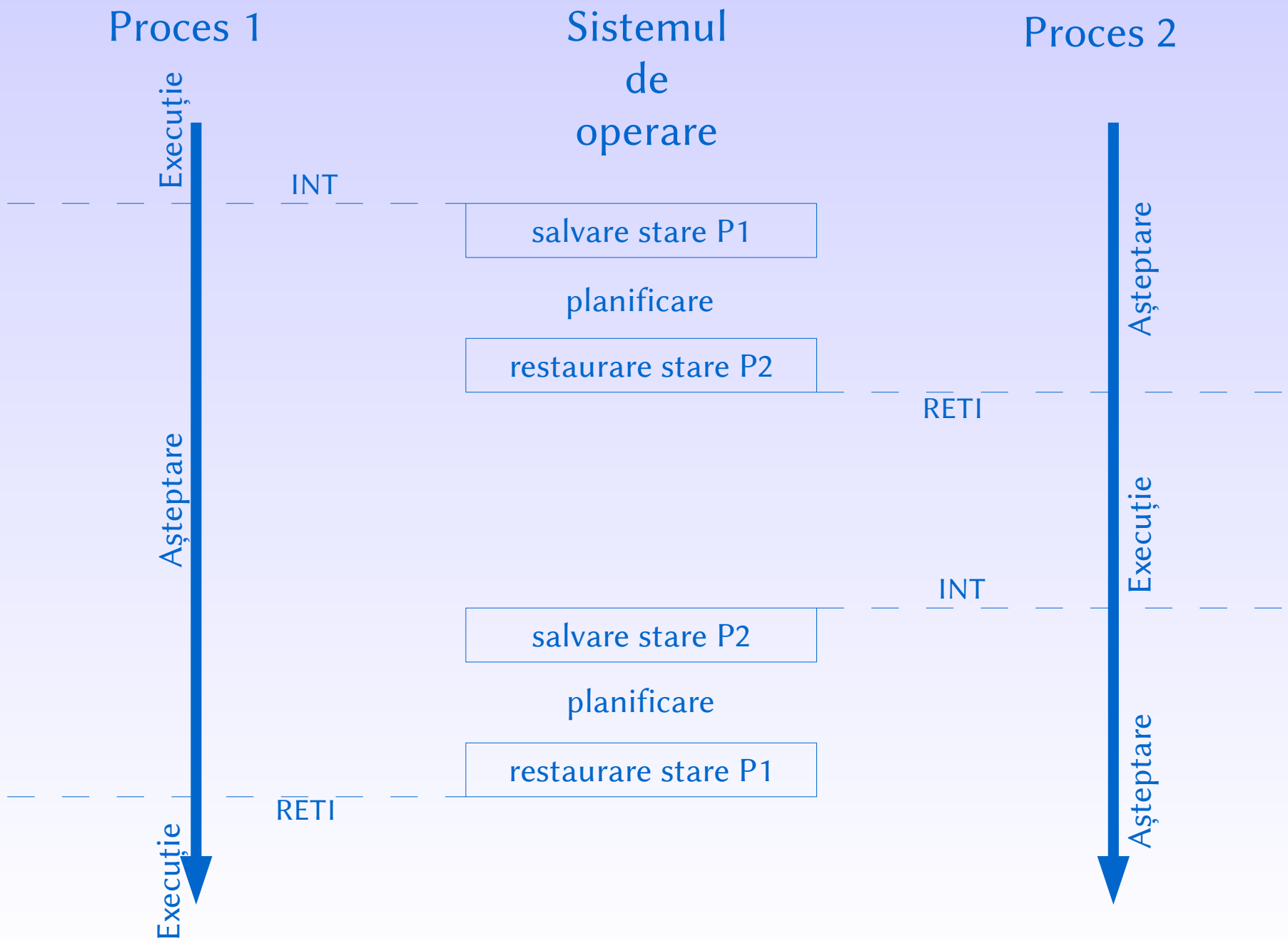
Structura unui program în execuție

- program:
  - stocat pe harddisk
  - codul programului (text)
  - variabilele inițializate (bss)
- execuție:
  - în memorie
  - variabilele locale + adrese de întoarcere (stivă)
  - variabilele dinamice (heap)
  - resurse CPU (registre de uz general, registre speciale, program counter)

2.1.1. Procese concurente

- fiecare proces deține propriul CPU virtual
- în realitate un singur CPU este comutat de la un proces la altul
- multitasking / multiprogramare
- program != proces
  - program: șablon static generator de procese
  - proces: program în execuție
  - poate exista mai multe procese pe baza aceluiași program, fiecare având resursele asociate independent unul de celălalt

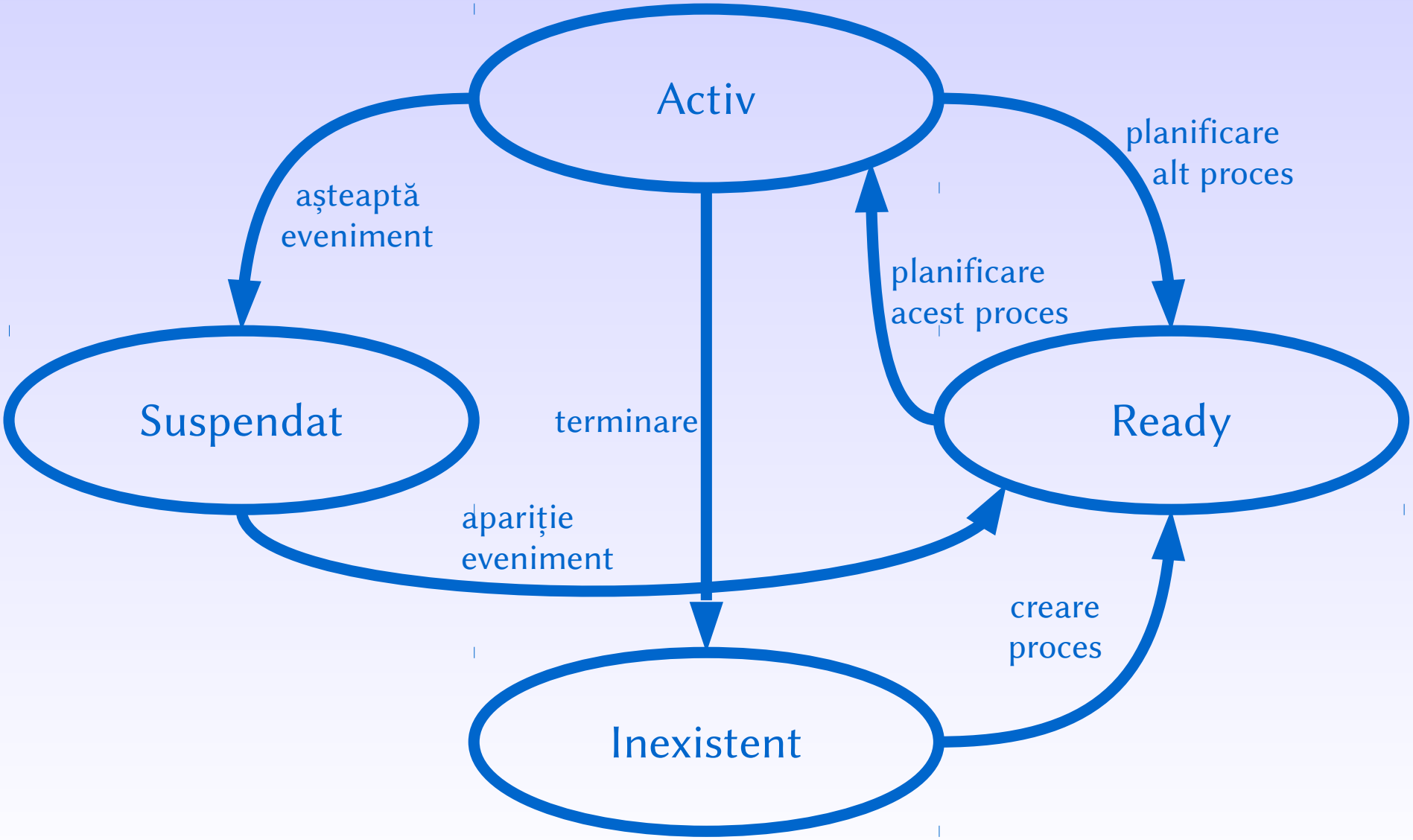
Rularea concurentă



Stările proceselor

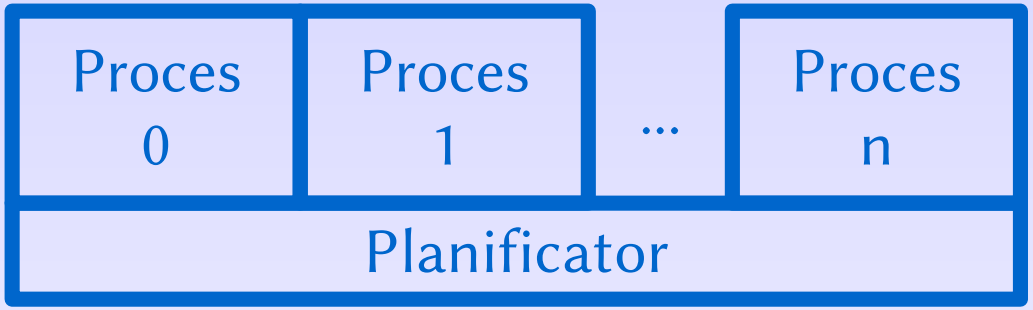
- activ
  - proces în execuție având controlul procesorului
- ready
  - are alocat toate resursele, în afara procesorului
- suspendat
  - așteaptă un eveniment, care să aducă în starea de a concura pentru procesor
- inexistent
  - un program ce nu rulează

Diagrama stărilor



Trecerea între stări

- model SO:



- nivelul inferior este planificatorul (scheduler)
- întreruperile, creările, activările și suspendările de procese sunt înglobate în scheduler
- restul SO este alcătuit din procese

2.1.2. Ierarhii de procese

- SO asigură mecanisme pentru crearea și distrugerea proceselor
- apelul sistem **`fork()`**
  - creează o copie identică a procesului apelant
  - cele două procese (părinte și fiu) continuă execuția în paralel
- părintele tuturor proceselor:
  - **init**
  - creează procese fiu care încarcă programele și le execută

2.1.3. Implementarea proceselor

- SO menține o tabelă (**process table**) cu intrare pentru fiecare proces
- fiecare intrare conține informații despre:
  - starea procesului
  - PC, SP, registre
  - alocarea memoriei
  - starea fișierelor deschise
  - informații de planificare
  - alte informații legate de comutarea Active $\leftrightarrow$ Ready

Exemplu de process table

Process management	Memory management	File management
Registre Program Counter Program Status Word Stack Pointer Process State Time of process started CPU time used Children's CPU time Time of next alarm Message queue pointers Pending signal bits Process id Flag bits	Pointer to text segment Pointer to data segment Pointer to bss segment Exit status Signal status Process id Parent process id Process group Real UID Effective UID Real GID Effective GID Bit maps for signals Flag bits	UMASK mask Root directory Working directory File descriptors Effective UID Effective GID System call preamble Flag bits

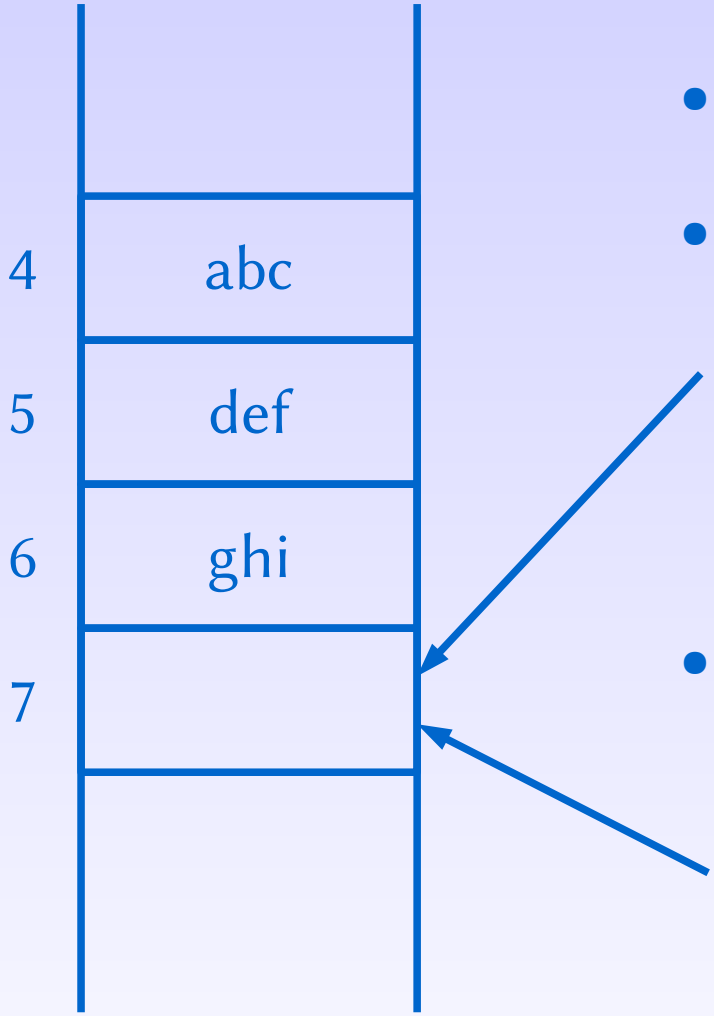
2.2. Comunicația între procese

- IPC – Inter Process Communication
- pentru rezolvarea diferitelor probleme comune, procesele trebuie să se comunice între ele
- comunicarea se poate face prin zone de memorie la care au acces amândouă procese
  - pipe
  - shared memory

2.2.1. Condiții de concurență

- condiții de concurență la accesul unor resurse
  - zone de memorie partajate
  - fișiere
- exemplu: print spooler
  - când un proces dorește să tipărească un fișier, depune numele fișierului într-un director special (printer spool)
  - există un proces (printer daemon) care verifică periodic conținutul directorului și dacă găsește ceva le tipărește și șterge din director

Accesul la printer spooler



- procesele A și B vor să tipărească
- se citește pointerul next_free_slot, stochează numele fișierului în acel loc și se incrementează pointerul
- dacă A e întrerupt între citirea next_free_slot și incrementarea acestuia, atunci cele două vor scrie numele fișierului în aceeași locație, și unul se va pierde

2.2.2. Secțiuni critice

- suprascrierea datelor în cazul unei condiții de concurență trebuie evitat
- e nevoie de excludere mutuală la accesul unei resurse comune
  - o cale de a se asigura că nici un proces nu va accesa o variabilă protejată în cazul în care un alt proces o utilizează
- secțiunea critică
  - partea dintr-un program unde se realizează accesul la o resursă comună

Condiții de cooperare corectă

- nu pot exista 2 procese simultan în secțiunea critică
- nu se poate face nici o presupunere asupra vitezei și numărului de CPU
- nici un proces care rulează cod în afara secțiunii critice nu poate bloca alte procese
- nici un proces nu va aștepta pentru totdeauna să intre în secțiunea critică

Dezactivare întreruperilor

- cea mai simplă soluție pentru realizarea secțiunilor critice
  - la intrarea în secțiune critică se dezactivează întreruperile, iar la ieșirea din secțiune se reactivează
  - planificatorul nu mai poate intervine în interiorul secțiunii critice
- procesul deține controlul asupra mașinii
  - poate duce la blocarea sistemului
- se folosește numai în interiorul kernelului

2.2.3. Excluderea mutuală prin Busy-Waiting

- busy-waiting: procesul așteaptă într-o buclă până la eliberarea resurselor pentru a intra în secțiunea critică
- implementare simplă
- consumă timp procesor

Lock variables

- se utilizează o singură variabilă partajată între procese
  - are valoarea 0 când nici un proces nu folosește resursele comune, și 1 când există un proces care folosește resursele comune
- la intrarea în secțiunea critică se testează această variabilă, dacă este 0 se intră în secțiune critică și se setează variabila la 1, altfel se așteaptă până când devine 0
- la ieșire se resetează variabila la 0
- există riscul ca cele două procese să intre simultan în secțiunea critică

Alternare strictă

Proces 0:

```
while (TRUE) {
    while (turn != 0); //wait
    critical_section();
    turn = 1;
    non_critical_section();
}
```

Proces 1:

```
while (TRUE) {
    while (turn != 1); //wait
    critical_section();
    turn = 0;
    non_critical_section();
}
```

Alternare strictă
dezavantaje

- Testarea continuă a unei variabile
  - trebuie evitată pentru că consumă timp procesor
- dacă unul din procese termină mult mai rapid atunci va trebuie să aștepte mult până când și celălalt termină atât secțiunea critică cât și cea non-critică
  - un proces în secțiunea non-critică blochează un alt proces: violarea condiției 3