

# 3

## Procese

11 martie 2009

- OSC
  - Capitolul 3 – Process Management
- MOS
  - Capitolul 2 – Processes and Threads
    - Secțiunea 1
- LSP
  - Capitolul 5 – Process Management
  - Capitolul 9 – Signals
- WSP
  - Capitolul 6 – Process Management
  - Capitolul 11 – Interprocess Communication

- Procese: definiție, relevanță
- Stări ale proceselor
- Planificarea proceselor
- Operații cu procese
- Procese în Linux și Windows
- Comunicația interproces

- Abstractizare de bază a SO împreună cu fișierul
  - abstractizare a mașinii fizice
  - abtractizează o acțiune
- Un program aflat în execuție
  - o instanță a unui program
- Program – entitate pasivă
- Proces – entitate activă
- Program = imaginea unui proces

- Programul este pasiv – este un executabil
- Procesul este activ – se rulează cod pe procesor
- Procesul are asociate resurse
  - spațiu de adresă
  - fișiere deschise
  - zone de memorie (cod, date, partajată)
- Care este asocierea program/proces?
- Ce zone de memorie aparțin doar procesului, nu și executabilului (sunt dinamice)?

```
$ file a.out
```

```
a.out: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), for GNU/Linux 2.6.8, dynamically linked  
(uses shared libs), not stripped
```

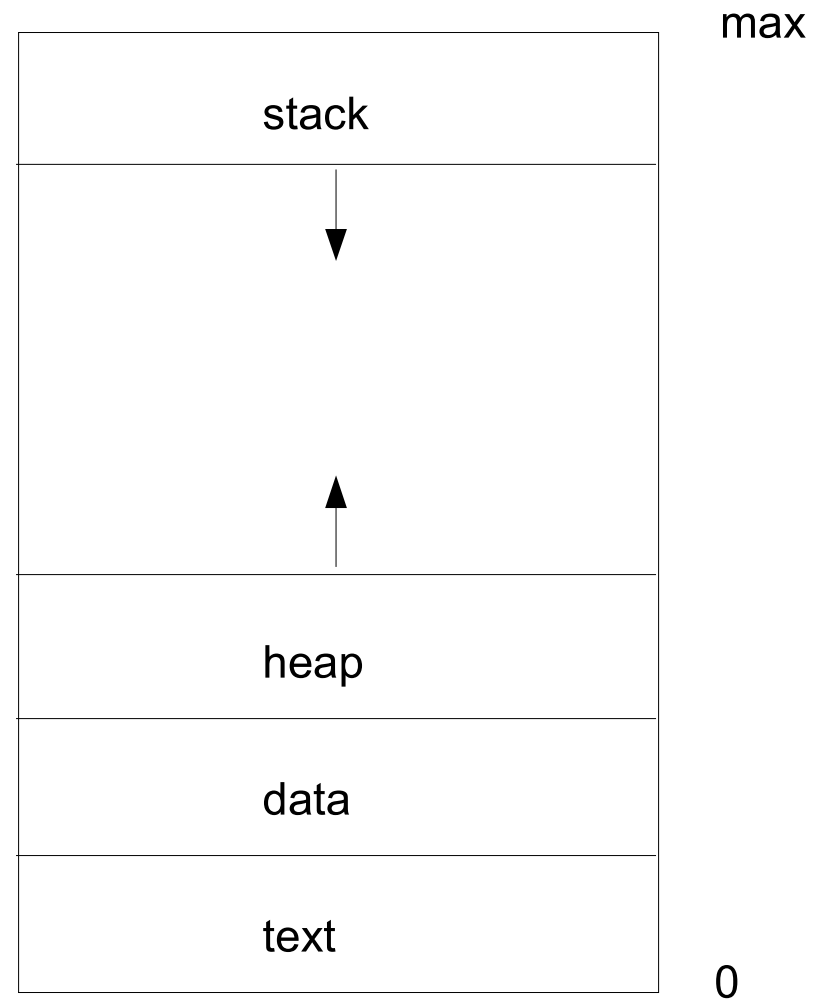
```
$ objdump -h a.out
```

```
[...]
```

```
11 .plt          00000030 00000000004003b0 00000000004003b0 000003b0 2**2  
  
CONTENTS, ALLOC, LOAD, READONLY, CODE  
12 .text        000001a8 00000000004003e0 00000000004003e0 000003e0 2**4  
  
CONTENTS, ALLOC, LOAD, READONLY, CODE  
13 .fini        0000000e 0000000000400588 0000000000400588 00000588 2**2  
  
CONTENTS, ALLOC, LOAD, READONLY, CODE  
14 .rodata      00000012 0000000000400598 0000000000400598 00000598 2**2  
  
CONTENTS, ALLOC, LOAD, READONLY, DATA
```

```
[...]
```

- `$ ./a.out`
- Se creează un proces
- Imaginea este executabilul `a.out`
- Procesul are asociate resurse și un spațiu de adresă



```

# pmap 1
1:  init [2]
0000000000400000      36K r-x--  /sbin/init
0000000000608000       4K rw---  /sbin/init
00000000019e3000    132K rw---  [ anon ]
00007f2e12ca0000       8K r-x--  /lib/libdl-2.7.so
00007f2e12ca2000   2048K ----- /lib/libdl-2.7.so
00007f2e12ea2000       8K rw---  /lib/libdl-2.7.so
00007f2e12ea4000   1320K r-x--  /lib/libc-2.7.so
00007f2e12fee000   2044K ----- /lib/libc-2.7.so
00007f2e131ed000      12K r----  /lib/libc-2.7.so
00007f2e131f0000       8K rw---  /lib/libc-2.7.so
00007f2e131f2000     20K rw---  [ anon ]
[...]
00007f2e1364d000    112K r-x--  /lib/ld-2.7.so
00007f2e13847000     12K rw---  [ anon ]
00007f2e13865000     12K rw---  [ anon ]
00007f2e13868000       8K rw---  /lib/ld-2.7.so
00007fff1b855000    84K rw---  [ stack ]
00007fff1b94b000       4K r-x--  [ anon ]
ffffffffffff600000    4K r-x--  [ anon ]
total                10316K

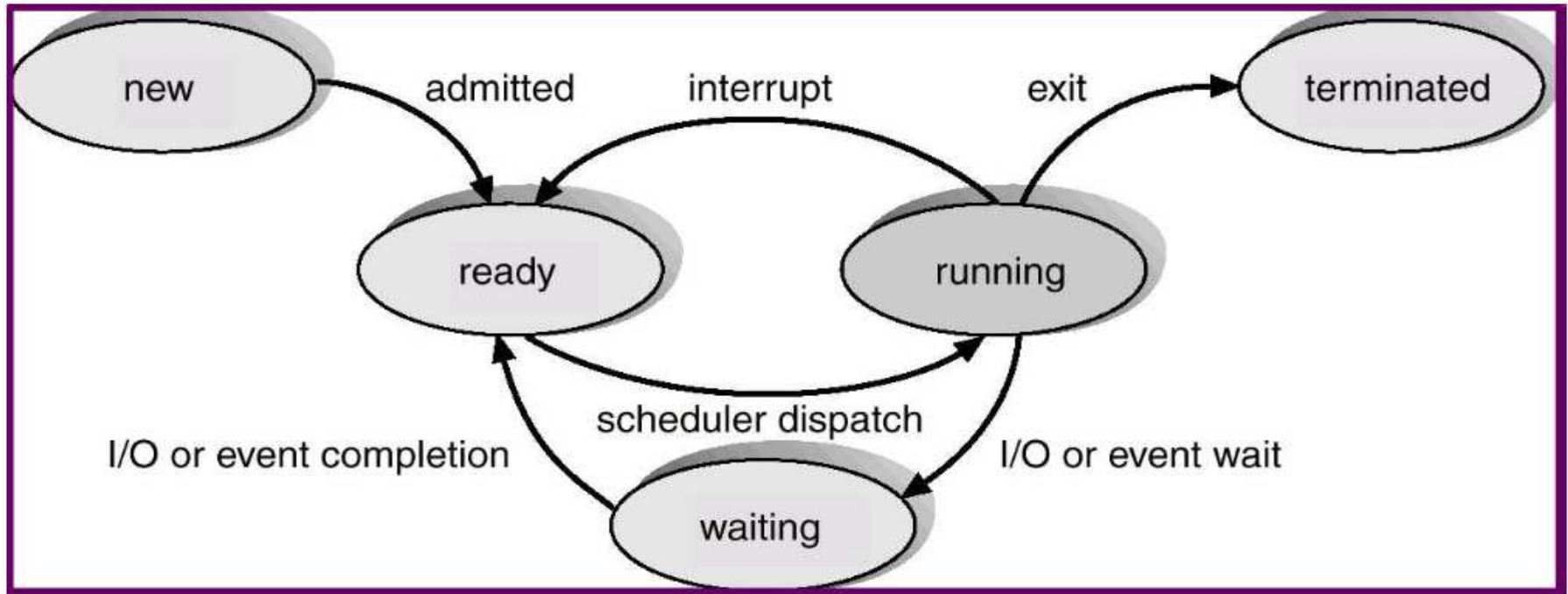
```



- Se înglobează o acțiune, o sarcină într-o structură independentă
- Coexistența mai multor procese
- Paralelism
  
- Câte procese pot rula simultan pe un sistem uniprocessor?
  - pseudo paralelism
  - paralelism efectiv (sisteme multiprocessor)
  
- Multiprogramare/multitasking

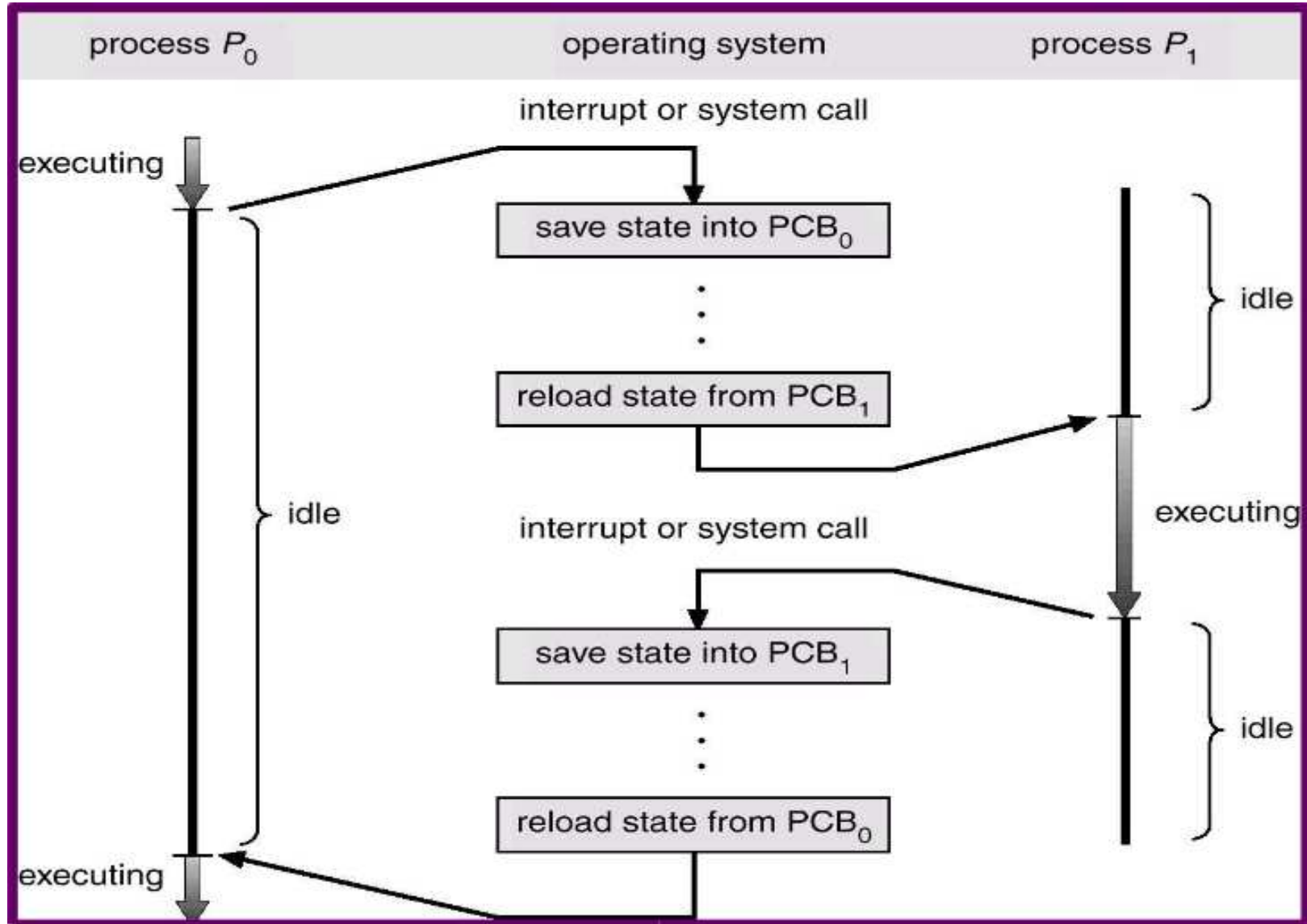
- Ce trebuie făcut?
- Cu ce resurse?
- Cum se realizează interacțiunea cu celelalte procese?

- Uniprogramare
  - avantaj: simplitate
  - dezavantaj: timpi morți, utilizare ineficientă a procesorului
- Multiprogramare
  - avantaj: folosirea eficientă a procesorului
  - dezavantaj: complexitate – algoritmi de planificare a proceselor pe procesoare
- Incertitudini ale multiprogramării
  - ordinea de execuție între două procese
  - durata de execuție a unui proces
  - durata de execuție a unei secțiuni a unui proces (sincronizare)



- READY – gata de execuție
- RUNNING – rulează
- WAITING – așteaptă încheierea unei operații de I/E

- Procese gata de execuție – coada READY
- Procese blocate – cozi de I/E
- Planificarea execuției (scheduling)
  - mod preemptiv
  - mod cooperativ



- Descrie procesul și resursele acestuia
  - în nucleul Linux – struct task\_struct
  - în nucleul Windows – EPROCESS
- PID (Process ID) – identificator numeric
- Spațiul de adresă
- Spațiul de nume (namespace); chroot
- Tabela de descriptori de fișier
- Masca de semnale
- Informații de securitate (user id; drepturile procesului)

- Subsistem al nucleului
- Planificarea proceselor pe procesoare
  - alegerea celui mai potrivit proces pentru a rula pe procesor
  - alegerea din coada READY
- Algoritmi și euristici pentru folosirea cât mai eficientă a procesorului
- Ține cont de natura proceselor:
  - CPU intensive
  - I/O intensive



- Resursele fizice sunt accesate prin resurse logice (abstractizare)
  - disc – fișiere, rețea – socket-uri
- Resursele pot avea asociată o stare
  - fișierele au cursor de fișier, un socket are o adresă și un port destinație
- Resursele pot fi
  - private (fișiere)
  - partajate (semafoare)
  - partajate parțial (spațiu de adresă)

- Fiecare resursă este adresată prin intermediul unor identificatori
  - Unix: descriptori (de fișiere, de IPC)
  - Windows: handle-uri
  - organizate de obicei în tabele; identificatorul este un index
- Folosirea unei resurse
  - obținerea/crearea unei resurse și alocarea unui descriptor
  - folosirea resursei cu ajutorul descriptorului și a funcțiilor oferite de SO
  - închiderea/distrugerea resursei

- Descriptorii sunt valabili doar în procesul în care au fost obținuți (process namespace)
- Într-un alt proces același descriptor poate identifica
  - o altă resursă
  - nici o resursă (descriptor invalid)
  - aceeași resursă (pentru resurse partajate)
- Duplicarea descriptorilor
  - resursa va fi partajată între descriptorul original și copie
- Crearea unui nou proces: partajare sau copiere?

- Procese interactive (foreground)
- Procese neinteractive (background)
  - trimise în background (operatorul &, comanda bg, SIGSTP)
  - daemoni (Unix): httpd, syslogd, sshd
  - servicii (Windows): IIS, print spooler
- Procese I/O intensive (I/O bound)
- Procese CPU intensive (CPU bound)

- Crearea proceselor
- Terminarea proceselor
  - voluntară (exit)
  - involuntară (excepții, erori fatale)
- Întreruperea proceselor
  - în Unix: semnale
  - în Windows: excepții
- Comunicare interproces
- Sincronizare

- Execuția unui program; rularea unei comenzi
- Un proces se creează dintr-un proces existent
- Relație părinte-copil
- Cine este procesul părinte în cazul execuției unei comenzi?
- Executabilul este imaginea noului proces
- Procesului  $i$  se asociază resurse
- Devine o entitate planificabilă pe procesor

```
PROCESS_INFORMATION pi;

if (!CreateProcess(NULL, "notepad", NULL, NULL, TRUE,
    NORMAL_PRIORITY_CLASS, NULL, NULL, &pi)) {
    printf("Cannot execute notepad.\n");
}

else {
    printf("notepad running\n");
    printf("notepad process handle is 0x%x\n",
        pi.hProcess);
}
```

- Creează un proces și execută programul transmis ca argument
- Controlul unui proces se face printr-un handle (ca și la fișiere)
- Pentru obținerea handle-ului
  - HANDLE GetCurrentProcess (VOID);
  - DWORD GetCurrentProcessId (VOID);
- Procesul părinte partajează handle-urile moștenibile cu procesul fiu



```
pid_t pid;

pid = fork();
switch (pid) {
case -1: /* fork failed */
    perror("fork");
    exit(EXIT_FAILURE);
case 0: /* procesul copil */
    ...
default: /* procesul părinte */
    printf("Created process with pid %d\n", pid);
    ...
}
```

- Se apelează o dată dar se întoarce de două ori
  - în procesul copil întoarce 0
  - în procesul părinte întoarce pid-ul procesului copil
  - de ce?
- Procesul copil este o copie a procesului părinte (spațiu de adresă)
- Resursele procesului părinte sunt partajate în procesul copil (fișiere, socketi)

- Pentru execuția unui program se înlocuiește imaginea curentă de executabil cu una nouă – apelurile din familia exec
- Se creează un spațiu nou de adrese
- Se încarcă executabilul
- PID-ul nu se schimbă
- Descriptorii de fișier sunt folosiți în continuare (doar dacă nu au flag-ul `CLOSE_ON_EXEC`)

- Când se termină un proces?
  - s-a încheiat programul (return exit\_code din main)
  - se programează terminarea procesului (exit/ExitProcess)
  - alt program termină procesul curent (kill/TerminateProcess)
- Încheierea forțată
  - Unix – semnale
  - Windows – excepții
  - abort - SIGABRT

- Un proces este întrerupt prin recepția unui semnal
- Când se primește un semnal:
  - starea procesului este salvată
  - se execută o rutină de tratare a semnalului
  - se continuă execuția procesului
- Sunt rezolvate asincron față de fluxul de execuție al procesului

- Sunt identificate prin numere
  - se recomandă folosire de macro-uri
  - SIGKILL – 9, SIGSEGV – 15
- Tipuri de semnale
  - asincrone – primite de la alte procese (programatic sau de la utilizator)
  - sincrone – procesul a generat o excepție (acces invalid, împărțire la 0, etc.)
- Ce se întâmplă la primirea unui semnal?
  - rularea unui handler
  - ignorarea semnalului
  - semnalul este blocat (fiecare proces are o mască de semnale)
  - comportament implicit (terminare sau ignorare) (SIGKILL și SIGSTOP nu pot fi ignorate)

- apelurile de sistem blocante pot fi întrerupte (errno = EINTR)
- lucrul cu semnale clasice este susceptibil la condiții de cursă
  - pot exista condiții de cursă dacă se partajează variabile între handler-ul de semnal și codul programului
- semnalele clasice pot fi pierdute
- nu se pot apela funcții non-reentrante (malloc)

- Numerele cuprinse între SIGRTMIN (32) și SIGRTMAX (63)
- Nu au roluri predefinite
- Se pot pune într-o coadă mai multe semnale
- Semnalele cu număr mai mic au prioritate mai mare
- Se garantează ordinea în cazul transmiterii repetate a aceluiași semnal



- Se datorează recepționării unei excepții
- Când un proces primește o excepție
  - starea procesului este salvată
  - se execută o rutină de tratare
  - se continuă execuția procesului
- Cauzele excepțiilor
  - o cerere de întrerupere a procesului
  - accesarea unei locații invalide de memorie
  - împărțire la zero

- Asynchronous Procedure Call
- Asociate per thread; fiecare thread are o coadă de APC
- APC-urile se execută în momentul în care thread-ul curent intră într-o stare alertabilă
  - WaitForSingleObjectEx

- Procesul apelant este blocat până la terminarea procesului specificat (proces copil)
  - Unix: wait, waitpid, WIFEXITED, WIFSIGNALED
  - Windows: WaitForSingleObject, GetExitCodeProcess
  
- Cum sunt implementați operatorii shell
  - & ;
  - || &&

- Ce se întâmplă dacă nu se așteaptă un proces?
- Ce este un proces zombie?
- Ce se întâmplă cu procesele ale căror părinte moare?

```
$ nohup btdownloadheadless --responsefile lin-  
kernel.torrent &> /dev/null &
```

- Comanda este pornită în background
- Procesul ignoră semnalul SIGHUP
- La încheierea unei sesiuni shell se transmite
  - SIGQUIT proceselor din foreground
  - SIGHUP proceselor din background
- Similar: disown (bash), dtach, screen

- Procese care rulează în background
- Nu au asociate un terminal de control
- Pot rula înainte de autentificarea unui utilizator
  - StartServiceCtrlDispatcher(...)
  - Daemon()
- Care este procesul părinte al daemon-ilor Unix?
- Cum interacționează utilizatorul cu un daemon?

- IPC
- Colaborare și competiție
- Colaborare – partajare/transmitere informație
- Competiție – acces exclusiv la resurse -> sincronizare
- Colaborare
  - fișiere
  - pipe-uri (anonime, cu nume – FIFO)
  - socket-uri
  - cozi de mesaje
  - memorie partajată

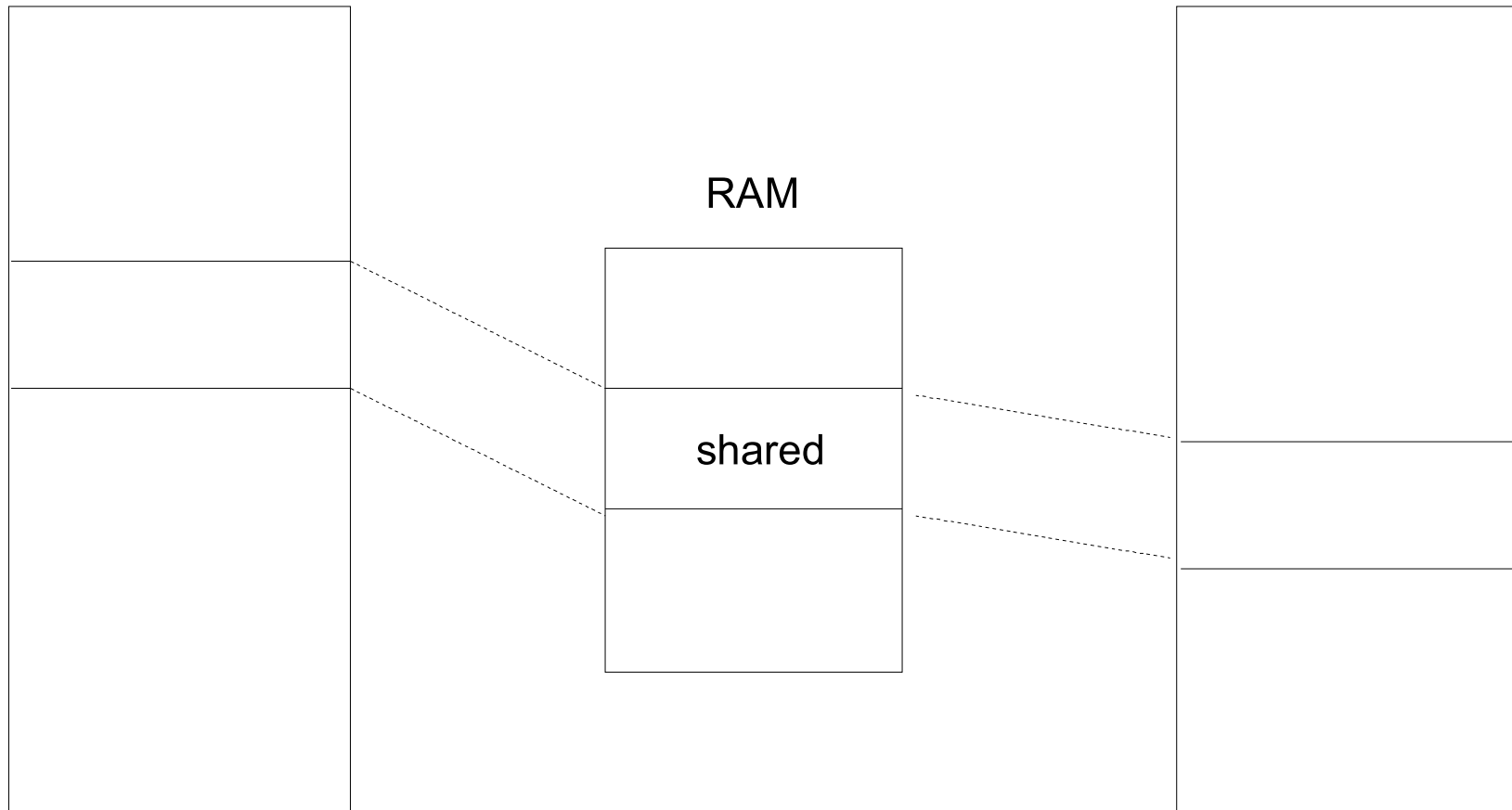
- Partajarea informației
  - acces concurent la informație -> sincronizare
- Paralelism/creșterea vitezei de calcul
  - modelul boss-workers, worker-threads
- Modularitate
  - Do one thing, do one thing well!
- Conveniență
  - un utilizator poate rula mai multe procese



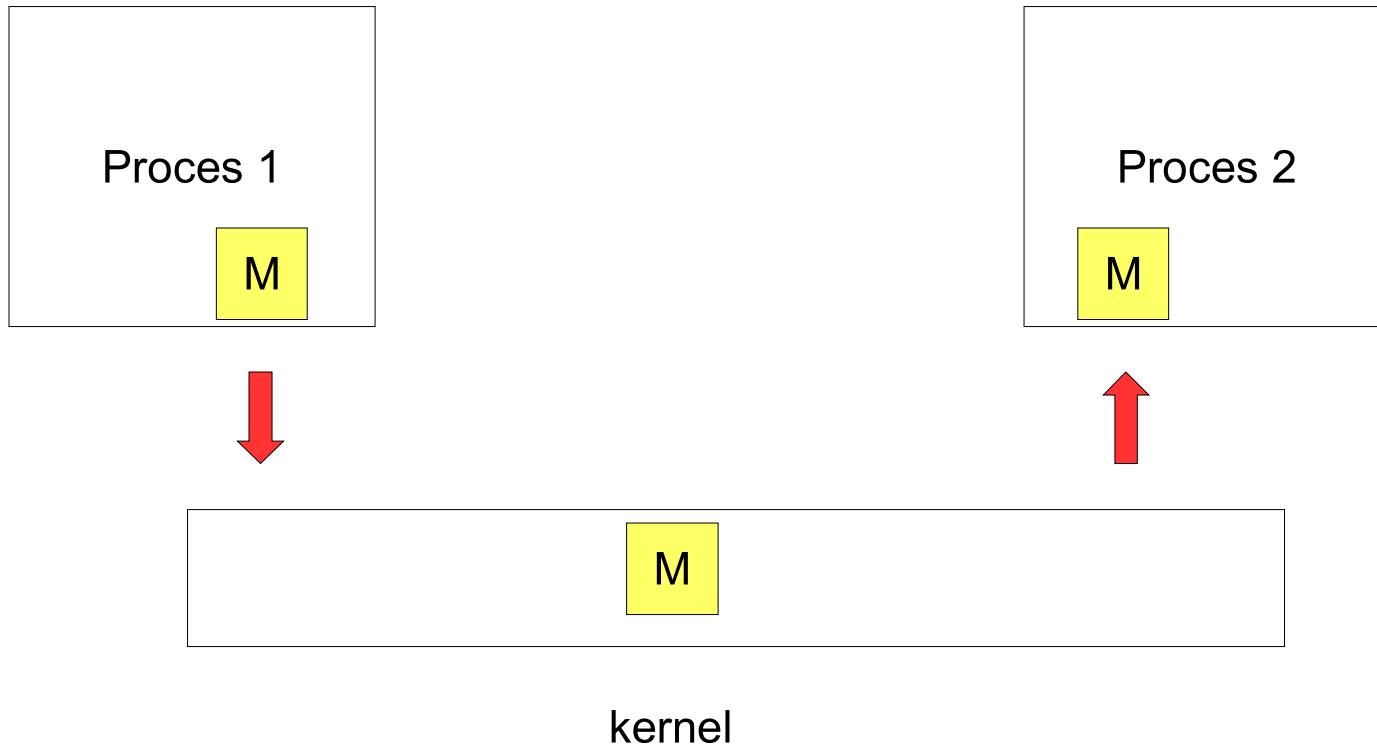
- Pipe-uri anonime
  - doi descriptori/două handle-uri
  - procesele comunicante sunt înrudite
  - un capăt de scriere și un capăt de citire
  - pipe/CreatePipe
  - Cum este implementat operatorul | în shell?
- Pipe-uri cu nume (FIFO)
  - intrare în sistemul de fișiere
  - procesele nu trebuie să fie înrudite
  - mkfifo/CreateNamedPipe

Spațiu adresă proces 1

Spațiu adresă proces 2

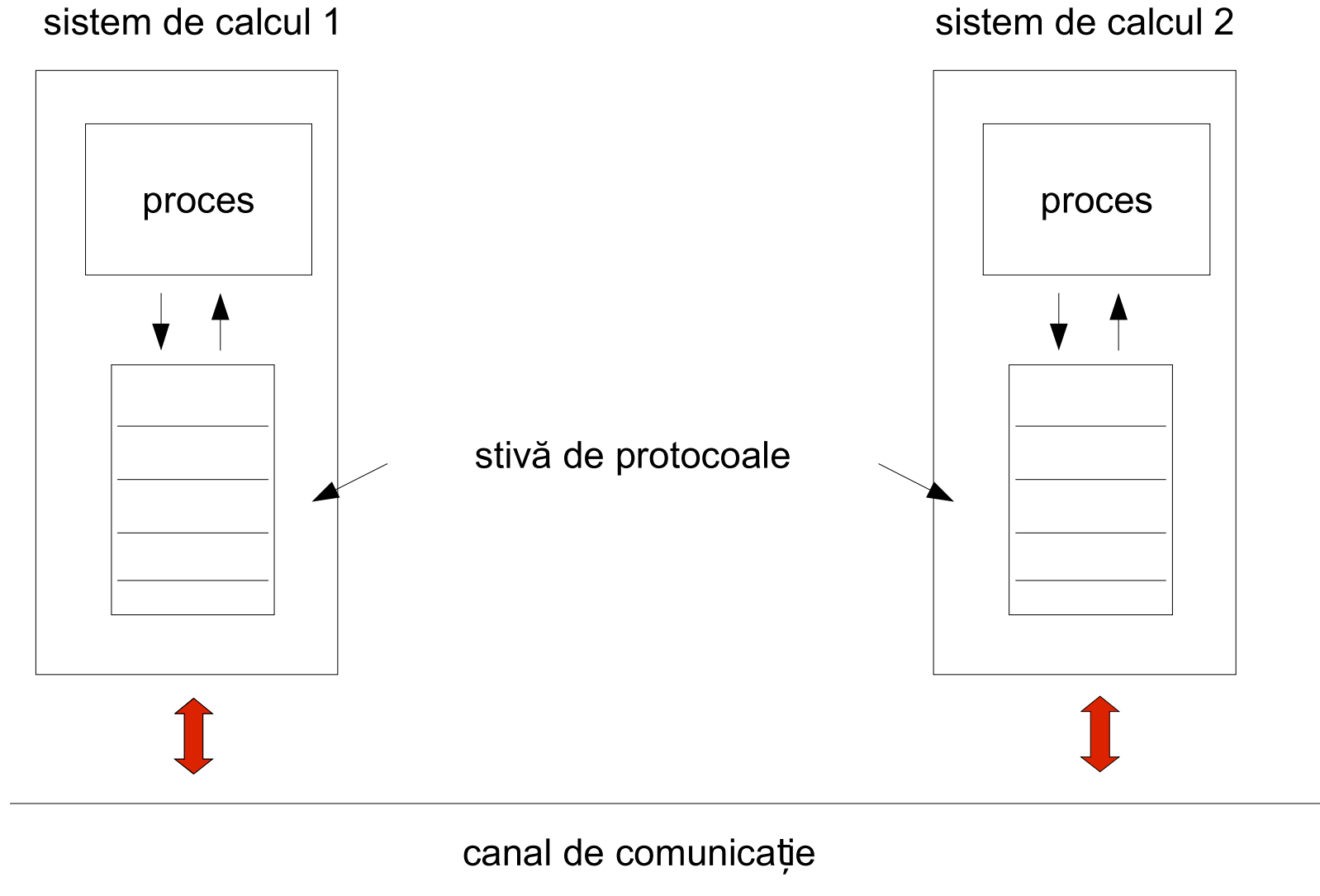


- Spațiu virtual de adresă mapat peste memoria fizică (RAM)
- Operații
  - crearea zonei de memorie partajată
  - atașarea la zona de memorie partajată
  - detașarea de la zona de memorie partajată
  - eliberarea zonei de memorie partajată
  - shmget, shmat, shmdt, shmctl
  - CreateFileMapping/MapViewOfFile/UnmapViewOfFile
- Accesul se face pe bază de pointeri (adrese)



- Util în medii distribuite (MPI)
- Tipuri de comunicație
  - dincronă sau asincronă
  - directă
    - mesajul este livrat procesului (MPI)
  - indirectă
    - mesajul este livrat într-o căsuță/mailslot (SO)
- Operații SO
  - creare mailslot (msgget/CreateMailslot, CreateFile)
  - primire/transmitere mesaj (msgsnd/msgrcv/ReadFile/WriteFile)
  - ștergere mailslot (msgctl/DeleteFile)

- Avantaje MP
  - util în sisteme distribuite
  - nu necesită sincronizare
  - informația este formatată (tip mesaj, lungime)
  
- Avantaje SM
  - util în sisteme multiprocesor
  - foarte rapidă (nu se face trecere prin kernel)
  - ușor de folosit (lucru cu pointeri/adrese)



- Abstractizare a comunicației în rețea
- Interfață de comunicație
- O pereche de procese comunicante în rețea – o pereche de socket
- Un socket = adresă + port
- Modelul client-server
- Comunicație
  - orientată conexiune (TCP)
  - ne-orientată conexiune (UDP)



- Remote Procedure Call
  - se oferă un stub clientului cu informații despre procedura de apelat
  - reprezentare independentă de sistem (XDR – eXternal Data Representation)
- Remote Method Invocation
  - object-based
  - se pot transmite obiecte ca argumente
  - stub (client) – parameter marshalling
  - skeleton (server) – parameter unmarshalling

- proces
- program
- executabil
- uniprogramare
- multiprogramare
- schimbare de context
- PCB
- scheduler
- descriptori de resurse
- creare, încheiere
- întrerupere
- semnale/APC
- daemoni/servicii
- shared memory
- message passing
- socket
- RPC, RMI

- Ce sunt procesele zombie și ce efect au asupra sistemului?
- Câte procese părinte (parent), fiu (child) și frate (sibling) poate avea un proces? (presupunând resurse nelimitate)
- Definiți trei caracteristici ale unui daemon/serviciu?

- Câte elemente de fiecare tip din lista celor de mai jos conține un proces:
  - executabil
  - stivă
  - handler de semnal
  - tabelă de fișiere
  - spațiu de adresă
  - descriptor de fișier

- Ce problemă poate aduce următorul program și cum poate fi soluționată?

```
static int flag = 0;
static void sig_handler (int signum)
{
    flag = 1;
}
int main (void)
{
    if (signal (SIGUSR1, sig_handler) == SIG_ERR) {
        perror ("signal");
        exit (EXIT_FAILURE);
    }
    /* wait for signal */
    if (flag == 0)
        pause ();
    /* do work */
    ...
    return 0;
}
```

