

Sisteme de operare

Procese

Sorin Milutinovici
sorinmilu@gmail.com

Procese

- Un proces este o instanță a unui program. Când solicitați sistemului de operare să ruleze un program, acesta este citit de pe sistemul de stocare, încărcat în memorie și apoi executat.
- În afara unor cazuri particulare, un program poate fi executat de mai multe ori, transformându-se în mai multe procese.
- Un proces poate lansa în execuție alte procese. Un proces poate avea mai multe subprocese dar un singur părinte. Dacă procesul nu are nici un părinte, înseamnă că a fost lansat în execuție de nucleul sistemului de operare.
- Pe sistemele de operare moderne, un proces poate avea mai multe fire de execuție (threads). Acestea sunt subprocese care împart aceleași resurse.
- Managementul proceselor reprezintă cea mai importantă funcție a unui sistem de operare.

Fiecare nucleu de procesor poate executa un singur proces

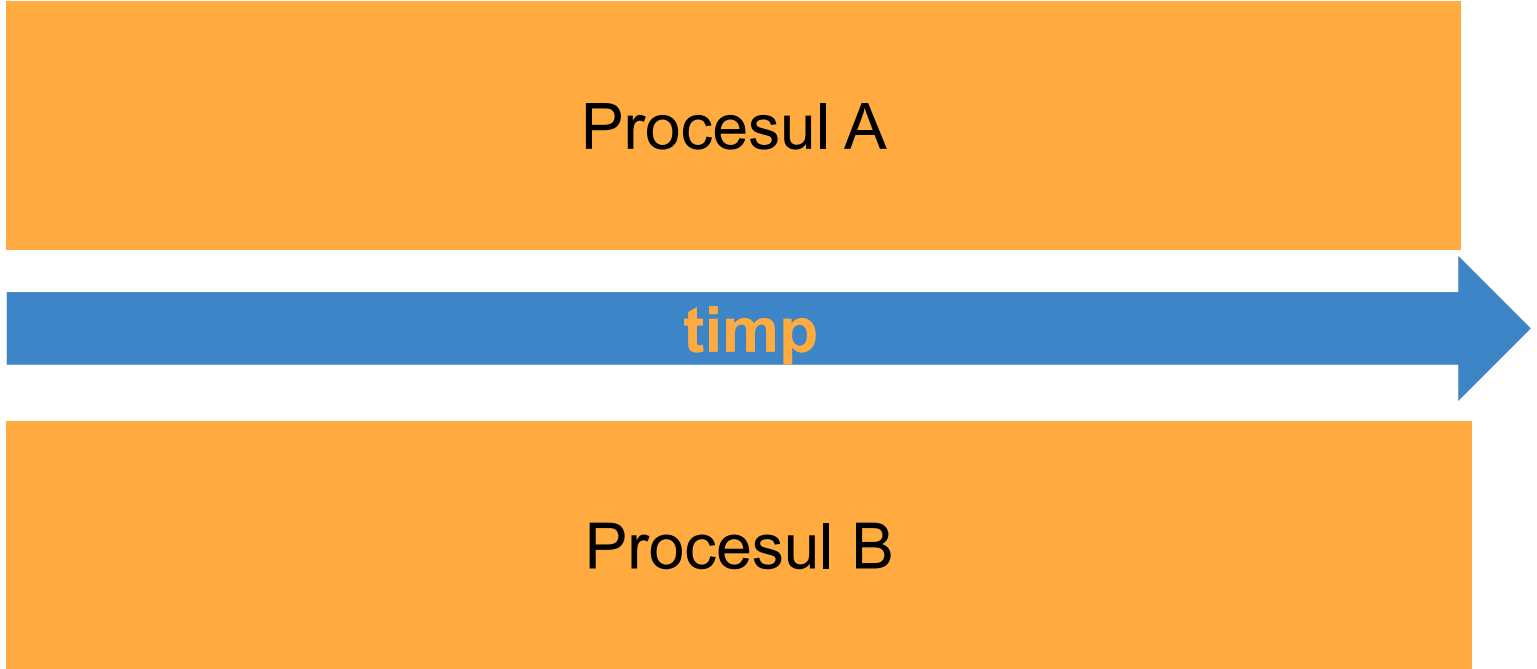
nucleu1

Procesul A

timp

nucleu2

Procesul B



Multitasking

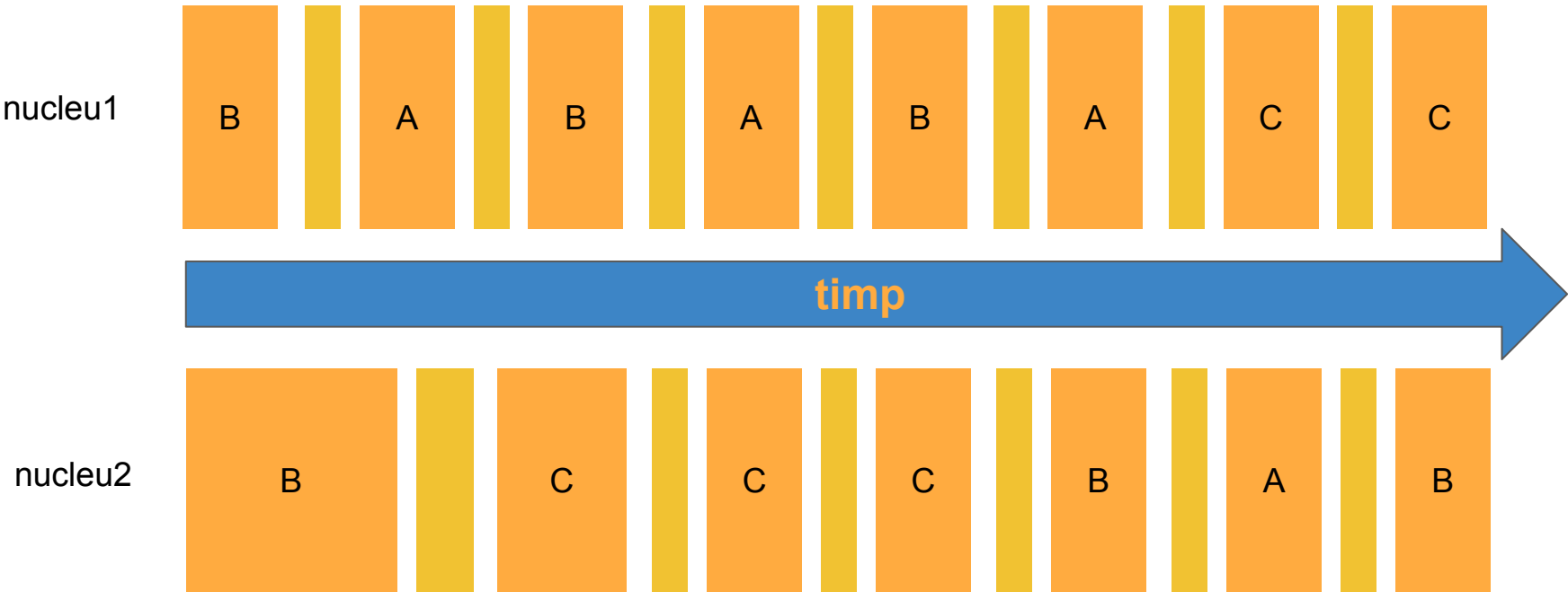
- Printre cele mai importante funcții ale unui Sistem de Operare este execuția mai multor procese în același timp, pe același nucleu (sau măcar o aproximație acceptabilă).
- Pentru a executa mai multe procese în același timp, procesorul trebuie să fie oprit din execuția unui program și să înceapă execuția unui alt program.
- Oprirea procesorului din execuția unui proces se face prin utilizarea întreruperilor hardware.
- **Întreruperile hardware** sunt semnale electrice care apar pe anumiți pini ai procesorului și care determină întreruperea procesului curent, înregistrarea stării acestuia și trecerea controlului procesorului către o subrutină specială numită *interrupt handler*.
- De câte ori apăsați pe o tastă, mișcați mouse-ul etc., se generează câte o întrerupere.

Multitasking

A, B, C = procese



cod al sistemului de operare



Multitasking

➤ Context switch

Sistemul trece de la rularea unui process la rularea codului nucleului sistemului de operare.

➤ Process switch

Sistemul trece de la rularea unui proces la rularea altui proces. Aceasta se face prin două context switch-uri (proces -> OS -> proces.)



Se întâlnește frecvent termenul de *context switch* folosit cu înțelesul de *process switch*.

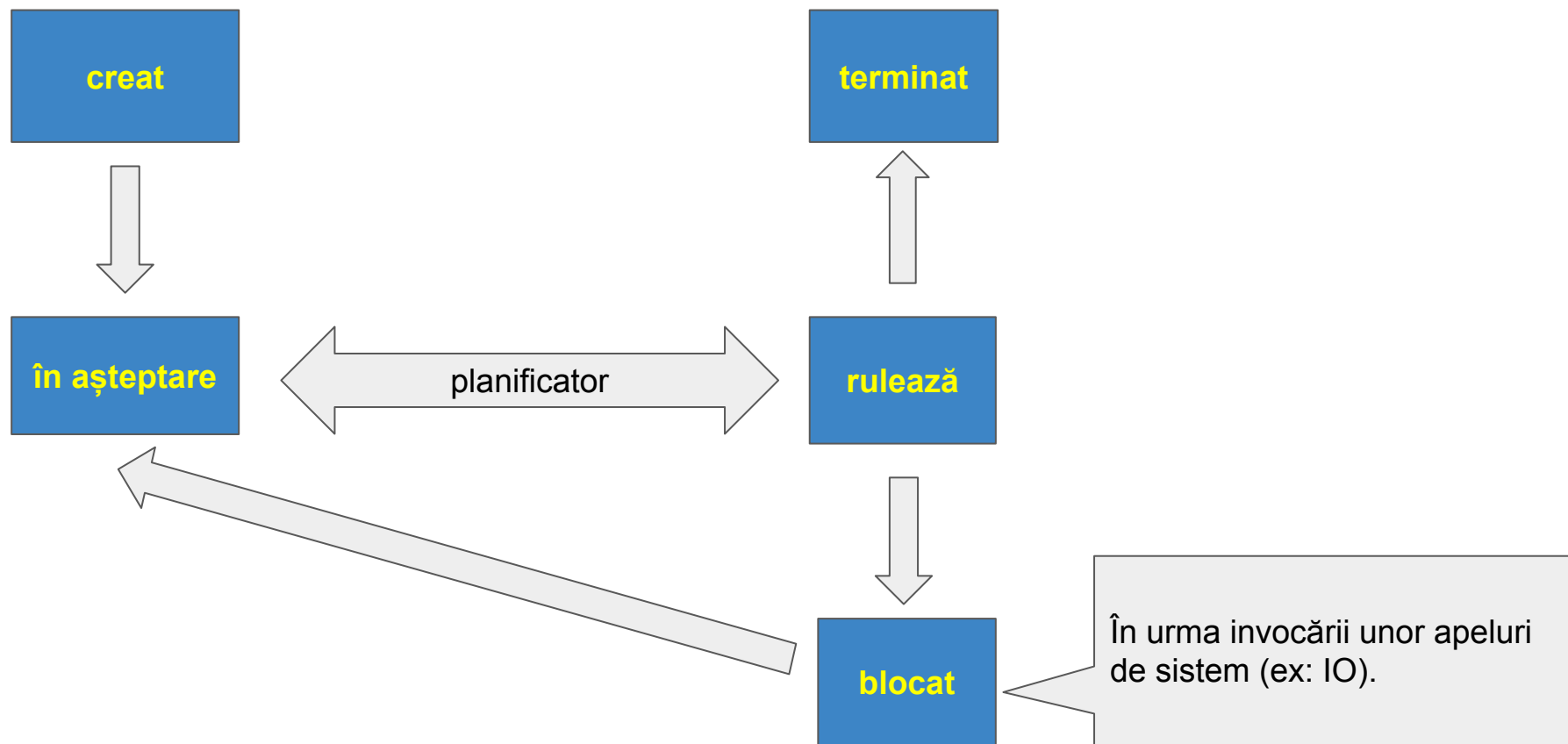
Multitasking

1. Procesorul primește o întrerupere
2. Întreruperea stochează numărul liniei curente a programului
3. Întreruperea invocă codul special asociaț acesteia (handler)
4. Handlerul salvează starea procesorului pentru proces
5. Handlerul invocă planificatorul.
6. Planificatorul selectează procesul care va rula în continuare
7. Planificatorul reface starea procesorului pentru acel proces selectat
8. Planificatorul se oprește și pornește execuția acelui proces.

Ce se întâmplă dacă nu se solicită nici o întrerupere?

- Toate plăcile de bază au un ceas care este setat să trimită întreruperi către procesor în mod regulat. Ceasul care întrerupe procesorul reprezintă numărul folosit pentru a evalua viteza procesorului, cu cât acesta e mai mare, cu atât procesorul este (teoretic) mai rapid.
- Operațiunea de “Overclocking” implică forțarea ceasului să trimită întreruperi mai des decât a decis fabricantul procesorului că acesta trebuie să le primească. Procesorul va funcționa mai repede dar se poate supraîncălzi.

Stările unui proces



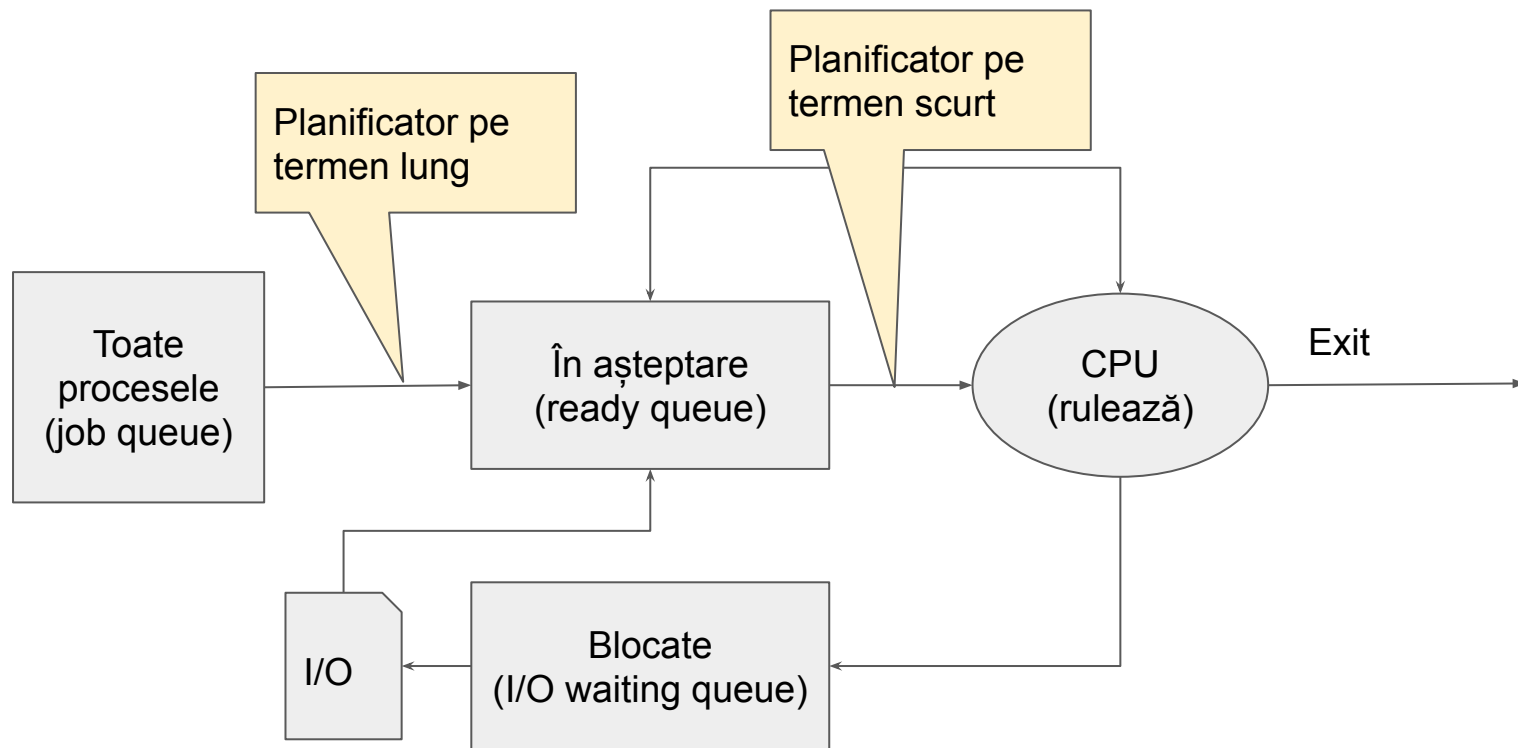
Planificator

Programe (module) care au ca scop alegerea procesului care va începe să ruleze după o întrerupere.

Există trei tipuri:

- **Planificator pe termen lung** (Job scheduler) - Mută procesele în “lista scurtă”
- **Planificator pe termen scurt** (CPU Scheduler) - Alege procesul care urmează să se execute din “Lista scurtă”. Când se vorbește despre scheduler, de regulă se vorbește despre acesta.
- **Planificator pe termen mediu** (Swapping scheduler) - Mută în memoria lentă (swap) procesele care așteaptă completarea unei operații de IO.

Liste de execuție (queues)



Pe lângă programarea timpilor de execuție, sistemul de operare trebuie să se ocupe și de managementul memoriei proceselor.

Planificator

Planificatoarele pot fi preemptive sau non preemptive.

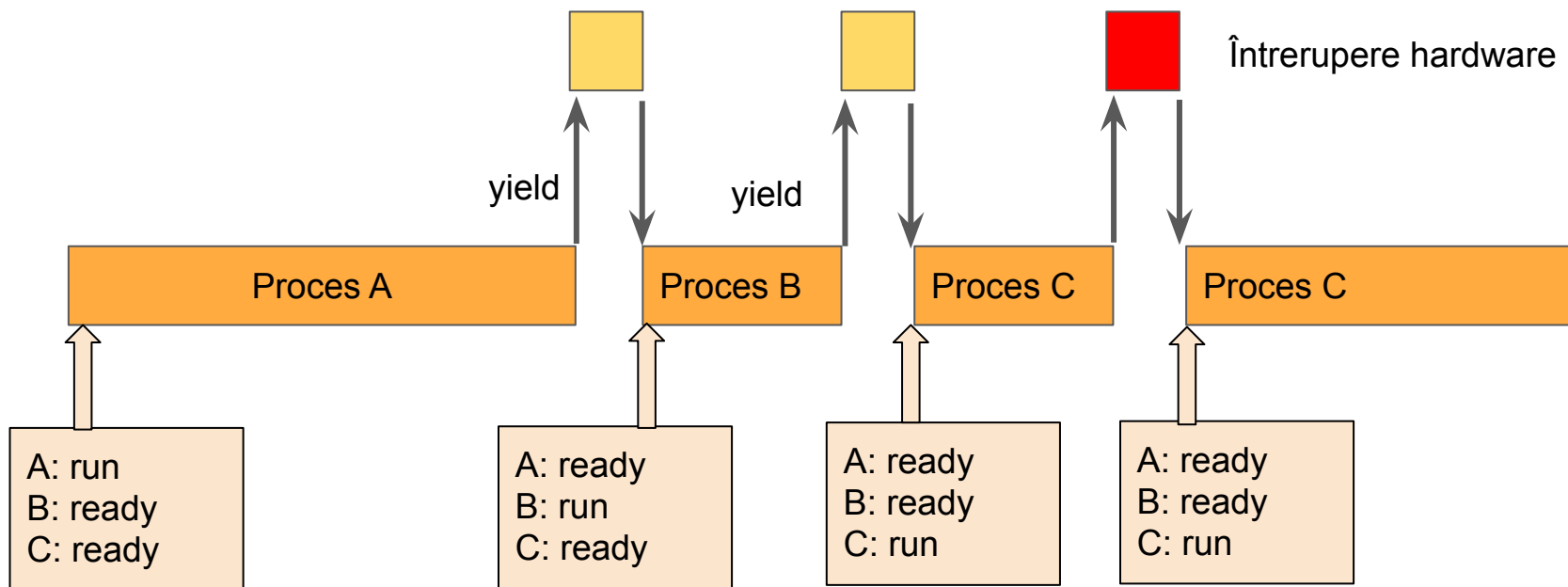
➤ ***Drept de preemțiune*** = privilegiu pe care îl are cineva, printr-un contract sau printr-o lege, la o vânzare-cumpărare, de a fi, în condiții egale, cel preferat dintre mai mulți cumpărători. (DEX online)

De multe ori, acționarii unei societăți pe acțiuni au drept de preemțiune la cumpărarea acțiunilor puse în vânzare de unul dintre ei.

Preemțiunea, în domeniul computerelor, înseamnă oprirea unui proces fără “acceptul” său pentru a da control altui proces.

➤ Toate sistemele de operare moderne cunoscute au multitasking preemptiv. Opusul acestuia este multitaskingul cooperativ - procesele cedează singure controlul

Planificator cooperativ



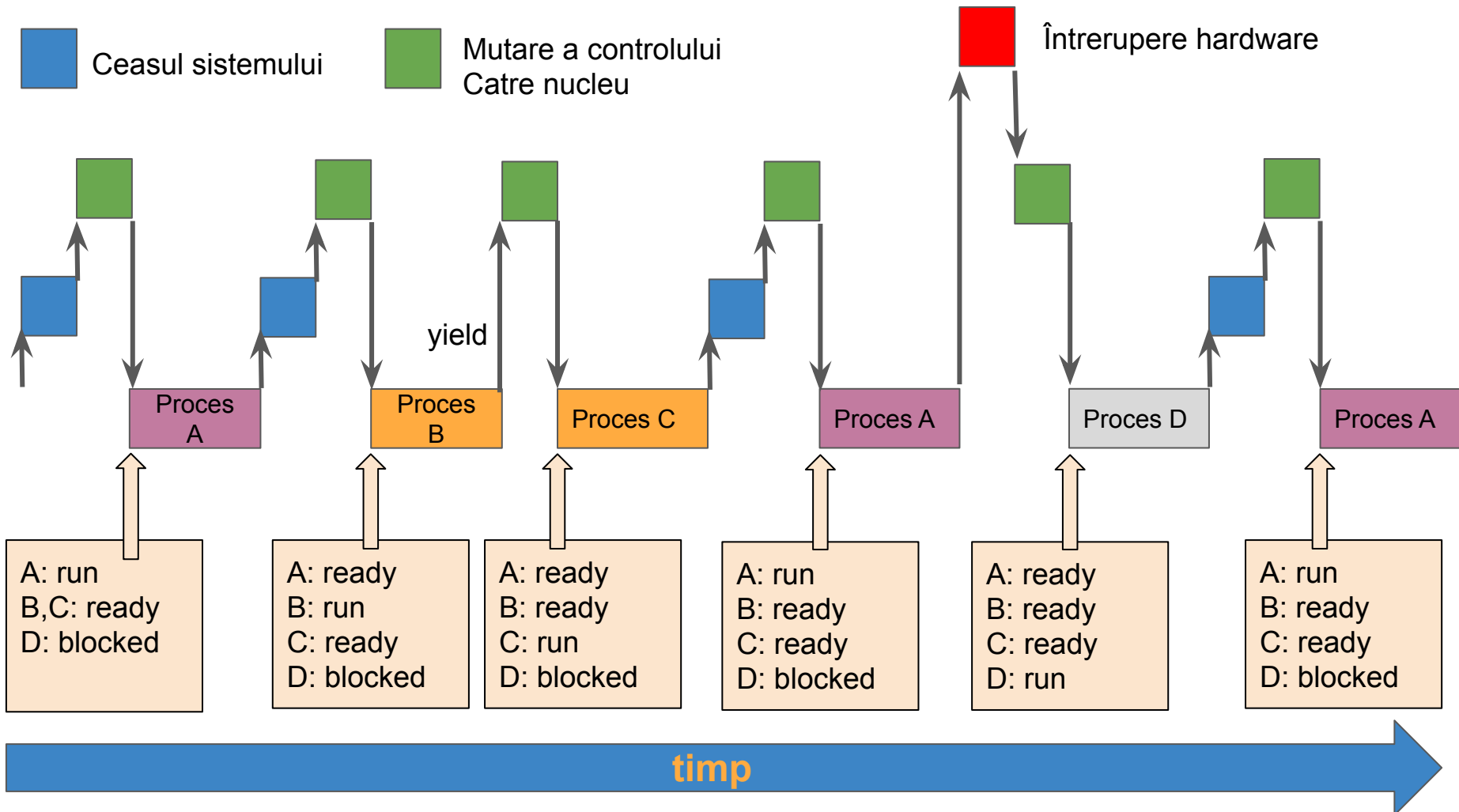
Windows pre 95



Mac OS clasic

Interactivitatea și eficiența sunt opuse. Tipuri de scheduler

Planificator preemptiv



Criterii de planificare

- **Utilizarea procesorului** - în mod ideal, procesorul trebuie să fie utilizat 100% din timp. Într-un sistem real, ocuparea procesorului variază între 40% și 90% (încarcare mare).
- **Tranzit (Throughput)** - Numărul de procese completate pe unitatea de timp. Poate să varieze între 10 pe secundă și 1 pe oră, în funcție de tipul procesului
- **Timpul de așteptare** - Timpul pe care procesele îl petrec în coada de așteptare. Exprimat uneori ca *încărcare medie* (**Load average**) Numărul mediu de procese care așteaptă să fie executate de procesor.
- **Timpul de răspuns** - Timpul mediu pe care un program interactiv îl consumă din momentul în care primește o comandă (click pe un buton) până când începe să ofere un răspuns.



În general se urmărește optimizarea unuia dintre criterii (ex: maximizarea utilizării procesorului). Dar aceasta va face sistemul să răspundă încet comenzilor.

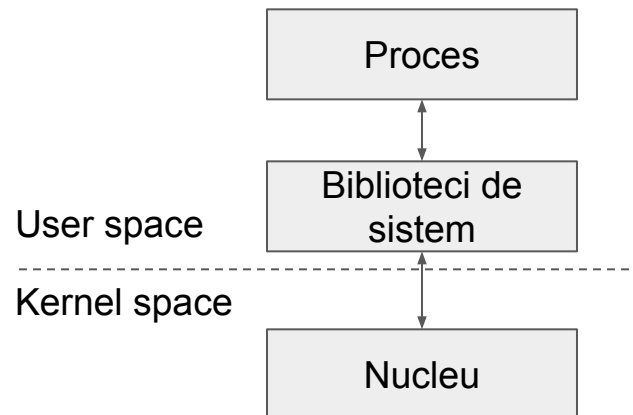
Algoritmi de planificare

- **First-Come First-Serve Scheduling, FCFS** - numai în multitasking cooperativ, o simplă coadă de execuție, primul venit primul servit (ca o coadă obișnuită)
- **Shortest-Job-First Scheduling, SJF** - sistemul încearcă să ghicească care proces va avea nevoie de cel mai scurt timp de execuție și îl alege.
- **Priority Scheduling** - O versiune îmbunătățită a SJF în care procesele au priorități diferite: cele mai importante vor fi executate primele. (ex: Solaris)
- **Multilevel Queue Scheduling** - Procesele sunt împărțite în mai multe cozi de execuție în funcție de prioritate. Fiecare coadă de execuție are (sau poate avea) propriul algoritm de planificare. Procesele nu se pot muta dintr-o coadă într-alta.
- **Multilevel Feedback-Queue Scheduling** - similar cu cel anterior doar că procesele pot fi mutate dintr-o coadă într-alta. (Windows NT, XP, Vista, Linux, Solaris)
- **Completely Fair Scheduler** - inspirat din algoritmi de planificare a traficului pe rețea, menține o serie de ponderi asociate fiecărui proces, ponderi care se recalculează frecvent și care determină o listă de priorități

Cum arată lumea prin ochii unui proces?

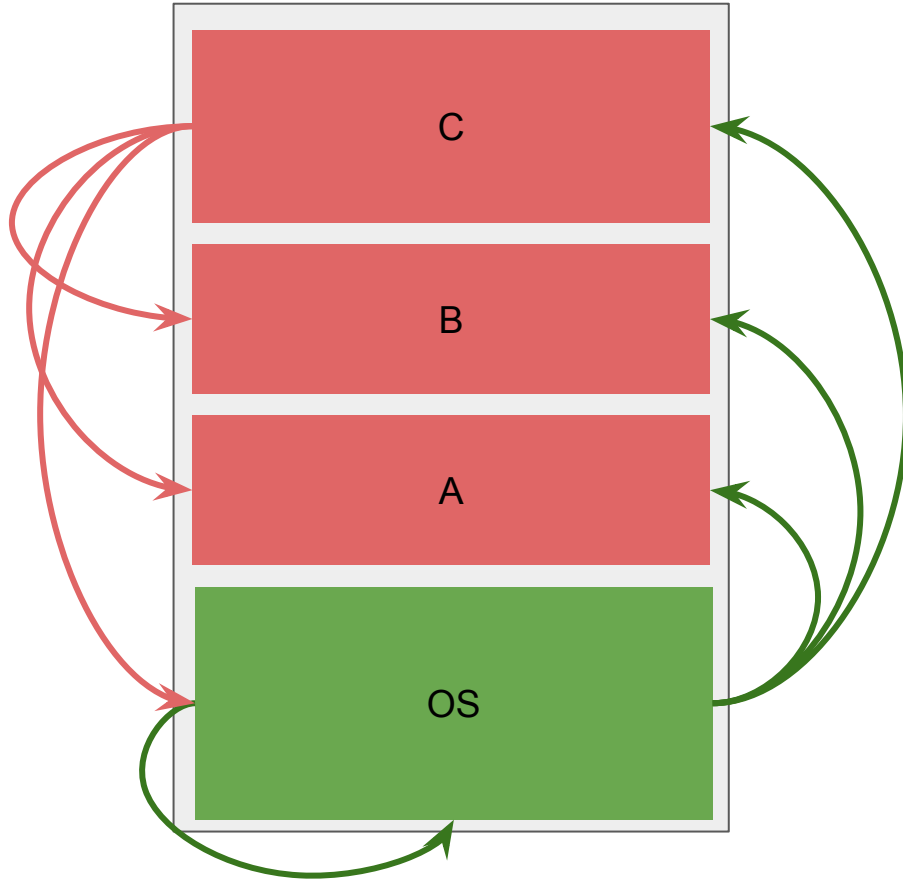
Fiecare proces “vede” propriul lui mediu

- Propriul spațiu de memorie
- Propriile fișiere
- Propriul procesor “virtual”



Pe lângă programarea timpilor de execuție, sistemul de operare trebuie să se ocupe și de managementul memoriei proceselor.

Fiecare proces are propriul spațiu de memorie



Sistemul de operare poate accesa spațiul de memorie alocat pentru orice proces.

Procesele nu pot accesa spațiul de memorie alocat pentru sistemul de operare.

Această restricție este realizată de hardware.

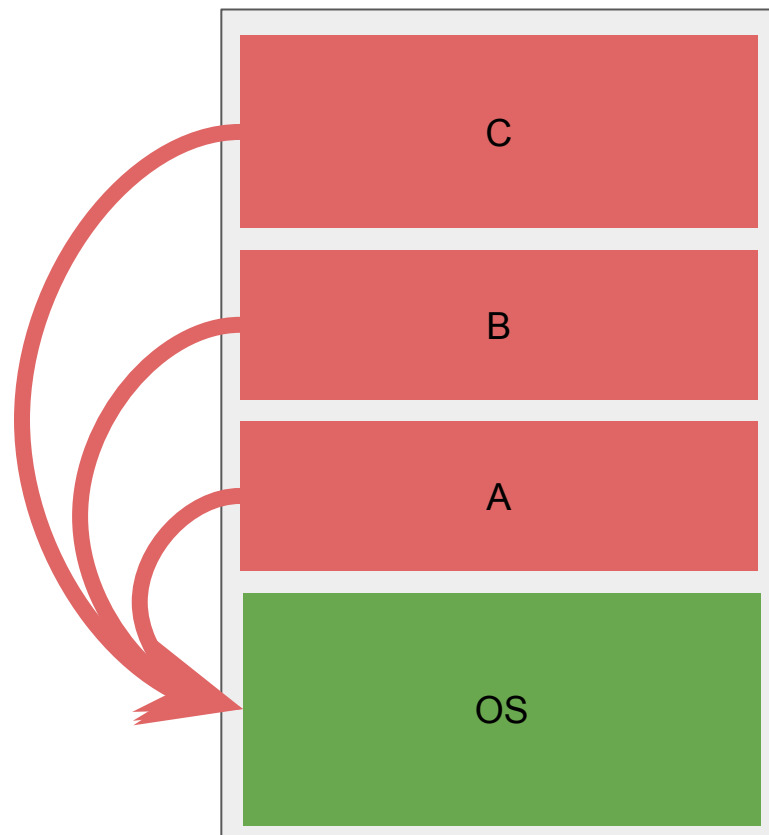
Dacă un proces încearcă să acceseze spațiul de memorie rezervat pentru sistemul de operare, programul se oprește și se emite eroarea "Segmentation Fault"

Procesele trebuie să interacționeze cu nucleul.

Apeluri de sistem (system calls)

- Un mecanism care permite proceselor să interacționeze cu codul nucleului sistemului de operare.
- Apelurile de sistem sunt stocate într-o tabelă și sunt apelate prin intermediul indicelui.
- În momentul apelului unui astfel de system call, execuția se mută la adresa de memorie stocată în tabelă.

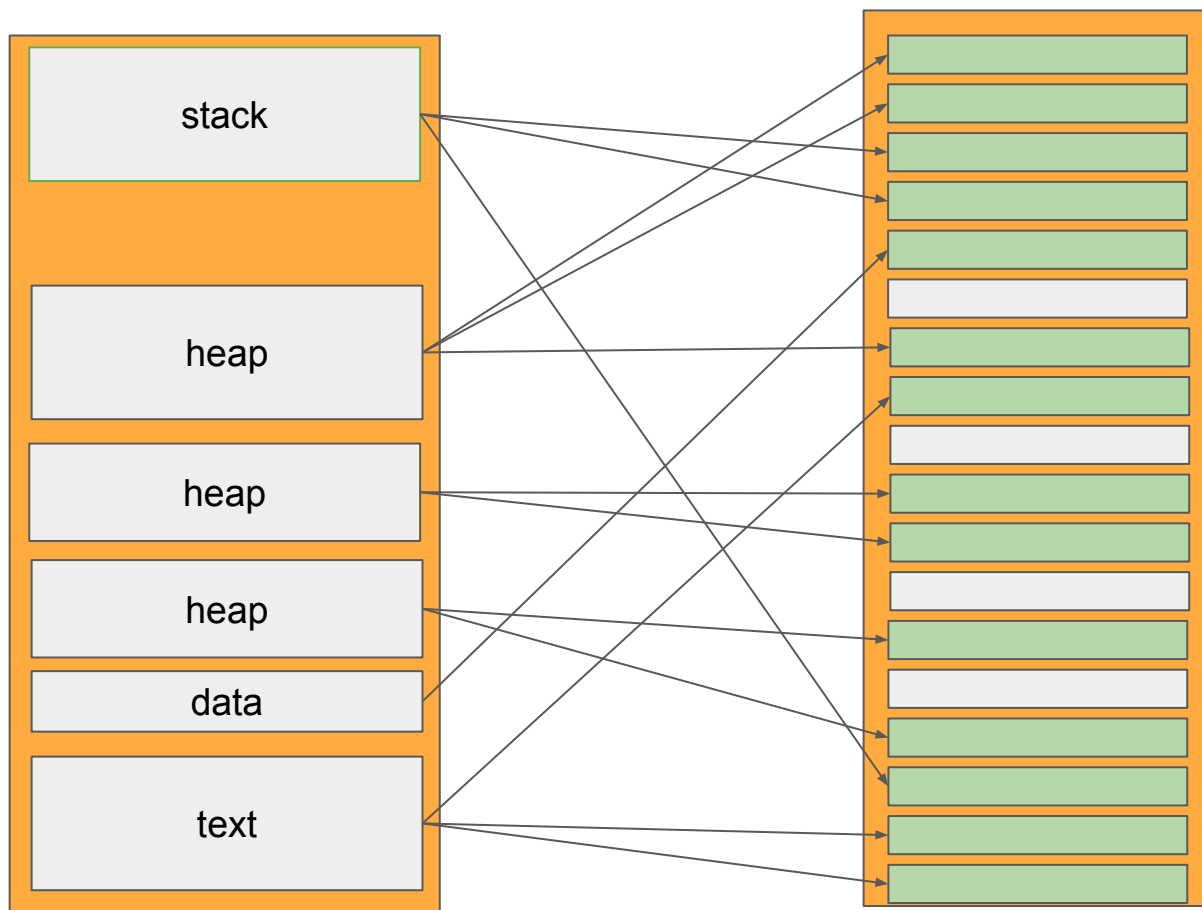
System call



Utilizarea memoriei

- Sistemul de operare păstrează câte un tabel de blocuri de memorie pentru fiecare proces.
- Sistemul poate să afle imediat care sunt spațiile de memorie atribuite fiecărui proces.
- De asemenea, sistemul poate verifica dacă procesele solicită adresarea unor blocuri de memorie care nu le aparțin.
- Dacă asta se întâmplă, sistemul de operare oprește execuția programului cu o excepție numită “Page Fault”.
- “Page Fault” provine de la faptul că blocurile de memorie sunt împărțite în fragmente numite “pagini”.

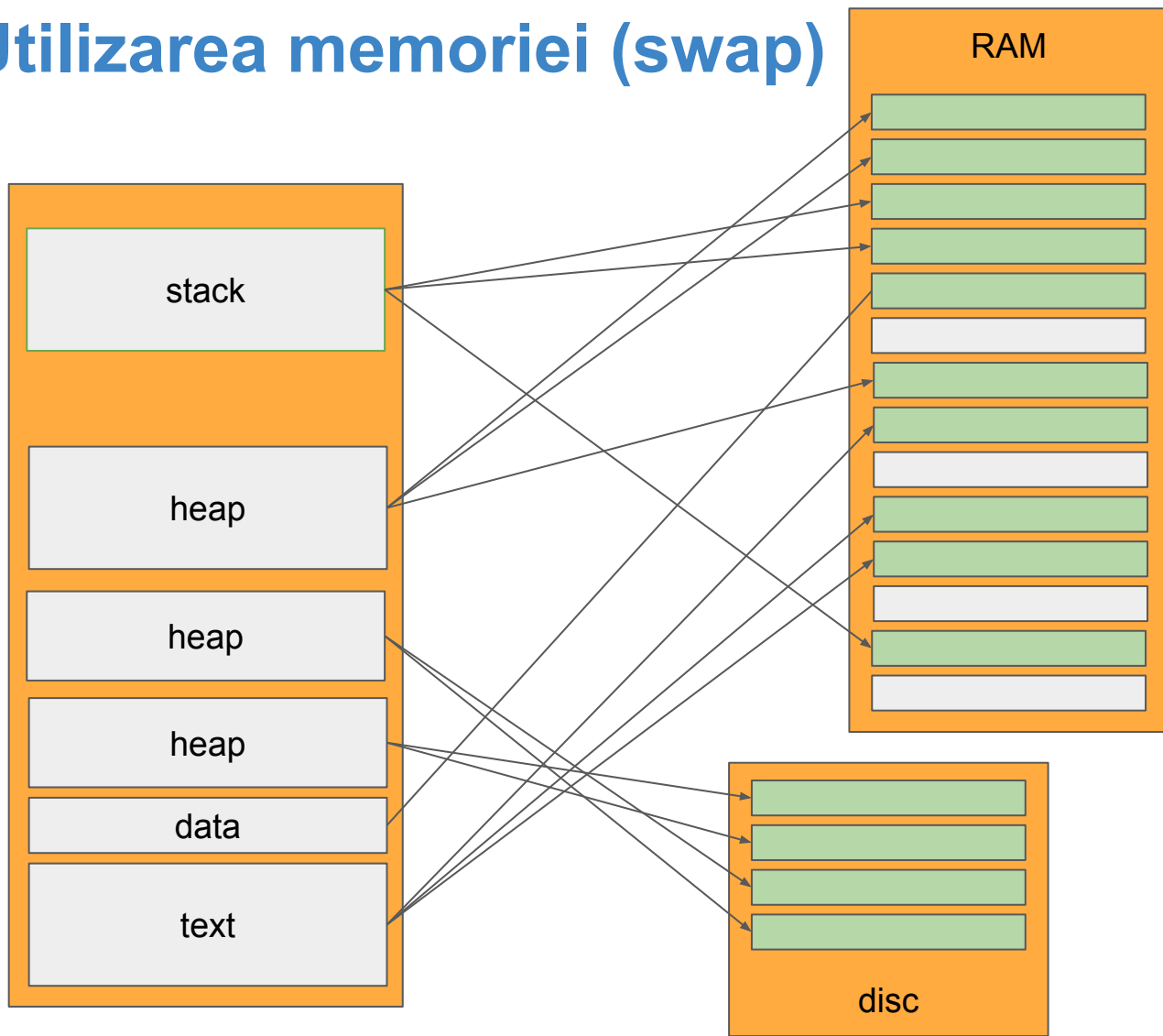
Utilizarea memoriei



Diferitele blocuri de memorie alocate unui proces sunt de fapt mapate unor pagini, nu neaparat în ordine.

Dimensiunea unei pagini depinde de arhitectură: în cazul procesoarelor x86 pe 32 de biti, aceasta este de 4Kb.

Utilizarea memoriei (swap)



Pentru a elibera memoria de lucru, sistemul de operare poate decide ca anumite blocuri să fie mutate pe un sistem de stocare mai lent (swap).

Dacă un proces încearcă să acceseze unul dintre blocurile marcate ca “swapped”, acesta va fi mai întâi copiat în memoria de lucru, modificandu-se tabela de adresare corespunzător.

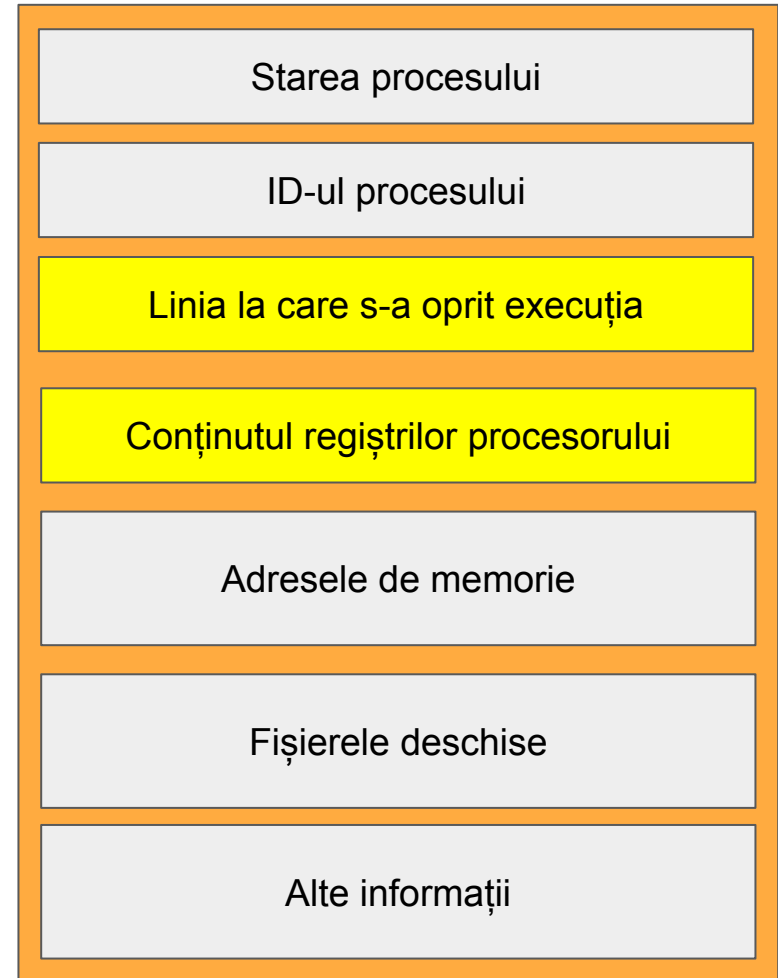
Swappingul permite proceselor să folosească mai multă memorie decât este disponibilă în sistem.

Memorie virtuală

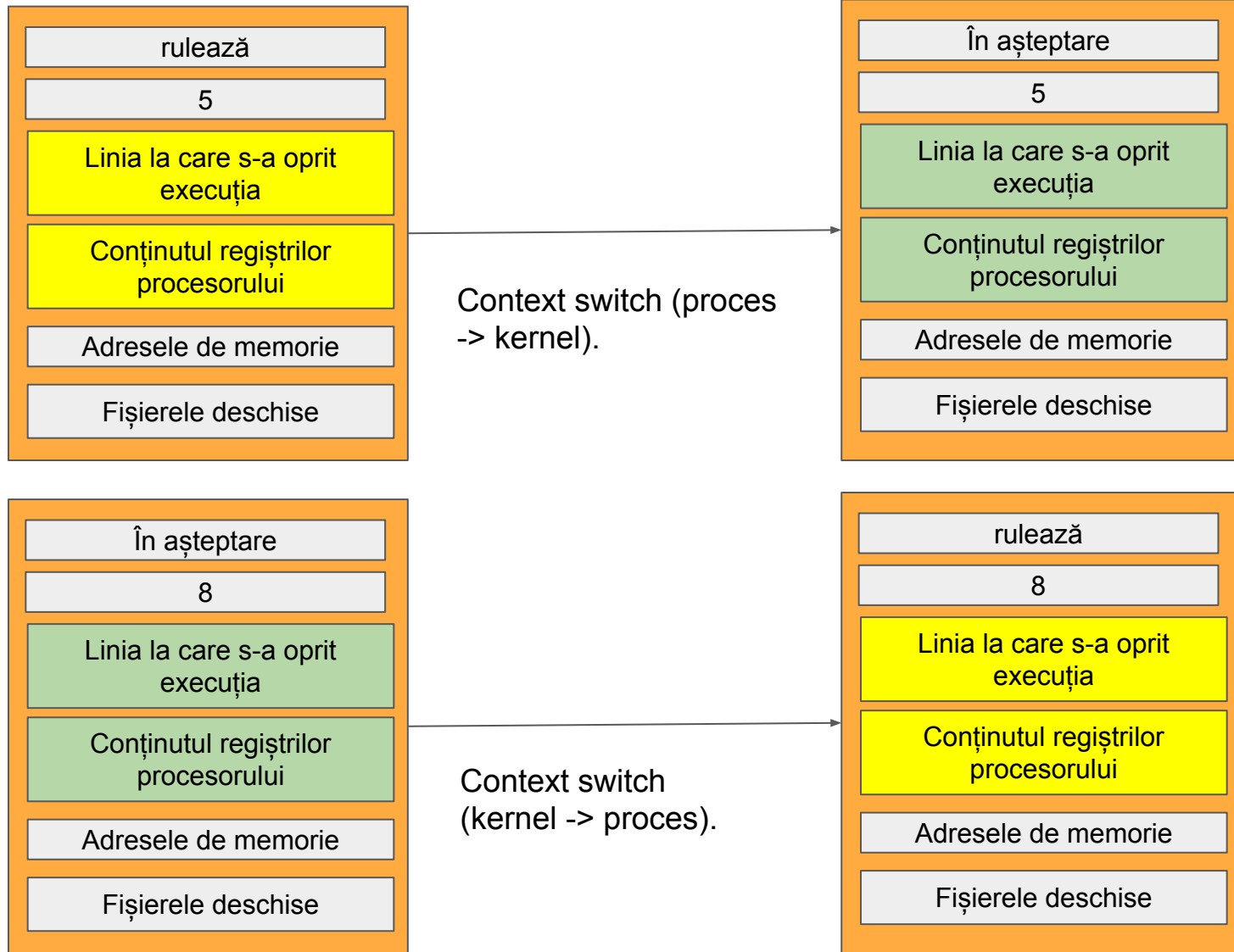
- Fiecare proces are acces la propriul spațiu de memorie pe care îl vede ca fiind continuu.
- În realitate, aceasta este o iluzie: memoria fiecărui proces este de fapt risipită sub formă de pagini separate în memoria RAM a sistemului.
- Maparea memoriei virtuale a fiecărui proces către adresele din memoria reală se face cu ajutorul unui component hardware numit MMU (Memory Management Unit).
- Procesele apelează funcții speciale în nucleu: `vmalloc()` sau `kmalloc()`. `Kmalloc()` alocă spații de memorie continue în RAM și se folosesc pentru I/O.
- `Kmalloc` are o limită (~4Mb în nucleele Linux moderne).

PCB

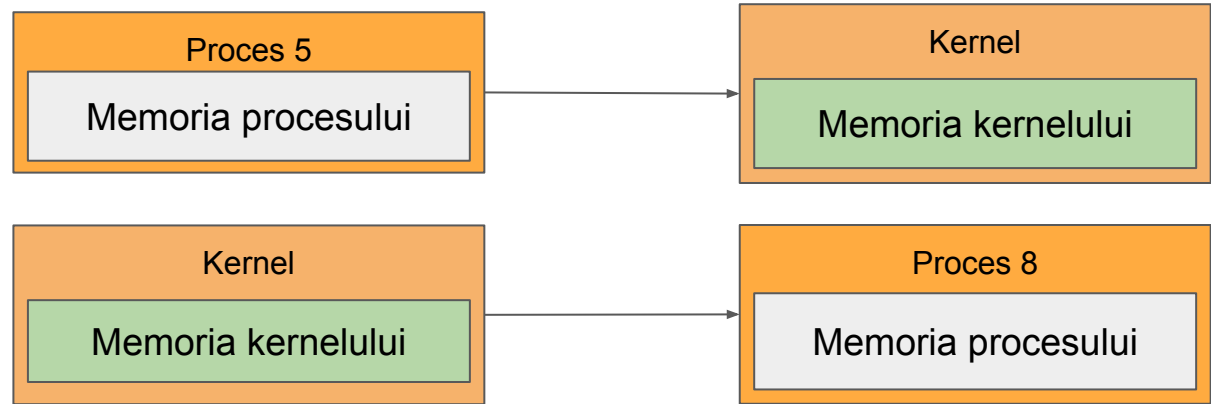
- La schimbarea stării, datele procesului se salvează în memorie.
- Sistemul conține o serie de containere unde poate salva datele unui proces atunci când acesta este oprit din rulare. Aceste date sunt stocate în structuri de date numite “process control block”.



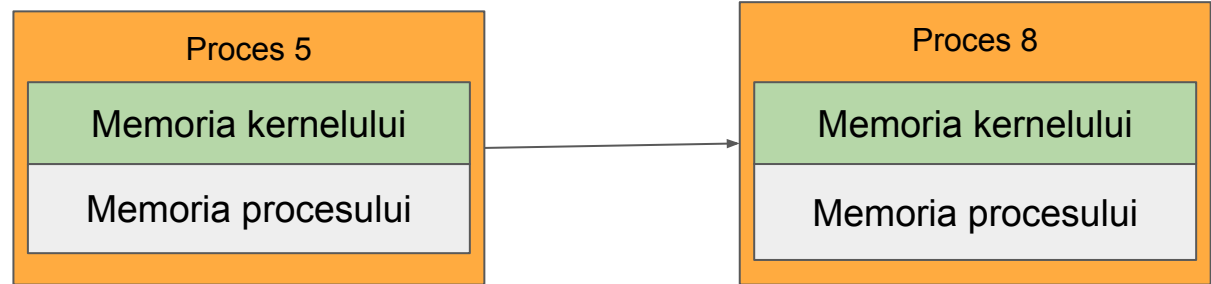
Mutarea între procese 5 și 8



Putem evita să facem două treceri și să putem muta direct între procesul 5 și procesul 8?



Putem dacă facem memoria kernelului să fie parte din memoria procesului.



Toate procesele care rulează pe un sistem de operare au o porțiune rezervată din spațiul de adrese în care se găsește kernelul sistemului de operare.

Procesoarele au implementate protecții hardware care interzic funcțiilor din spațiul de memorie al procesului să adreseze spațiul de memorie al nucleului direct

Mutarea
intre
procese



Efectul final este posibilitatea de a porni și rula mai multe aplicații în același timp (cel puțin aparent).

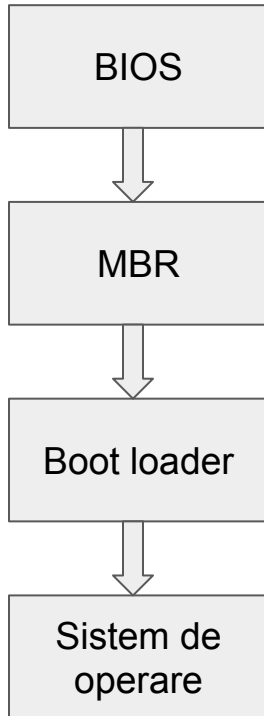
```
522
523 #if 0
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552 if (argc > 1) {
553     argv[0];
554     exit(2);
555 }
556
557 kbuild_verbose = getenv("KBUILD_VERBOSE");
558 if (kbuild_verbose)
559     verbose = atoi(kbuild_verbose);
560
561 filename = argv[1];
562 outputname = argv[2];
563 headername = argv[3];
564
565 fd = open(filename, O_RDONLY);
566 if (fd < 0) {
567     perror(filename);
568     exit(1);
569 }
570
571 if (fstat(fd, &st) < 0) {
572     perror(filename);
```

Anacron job 'cron.daily' on isp.content...

- Alerta - DIGI IPO (2 documente)
- Undelivered Mail Returned to Sender
- ** PROBLEM Service Alert: bigboy/N...
- ** PROBLEM Service Alert: bigboy/N...
- ** RECOVERY Service Alert: isp_conte...
- ** PROBLEM Service Alert: isp_conte...
- Alerta - Alerta Veolia/ApaNova (2 do...
- ** PROBLEM Service Alert: bigboy/N...

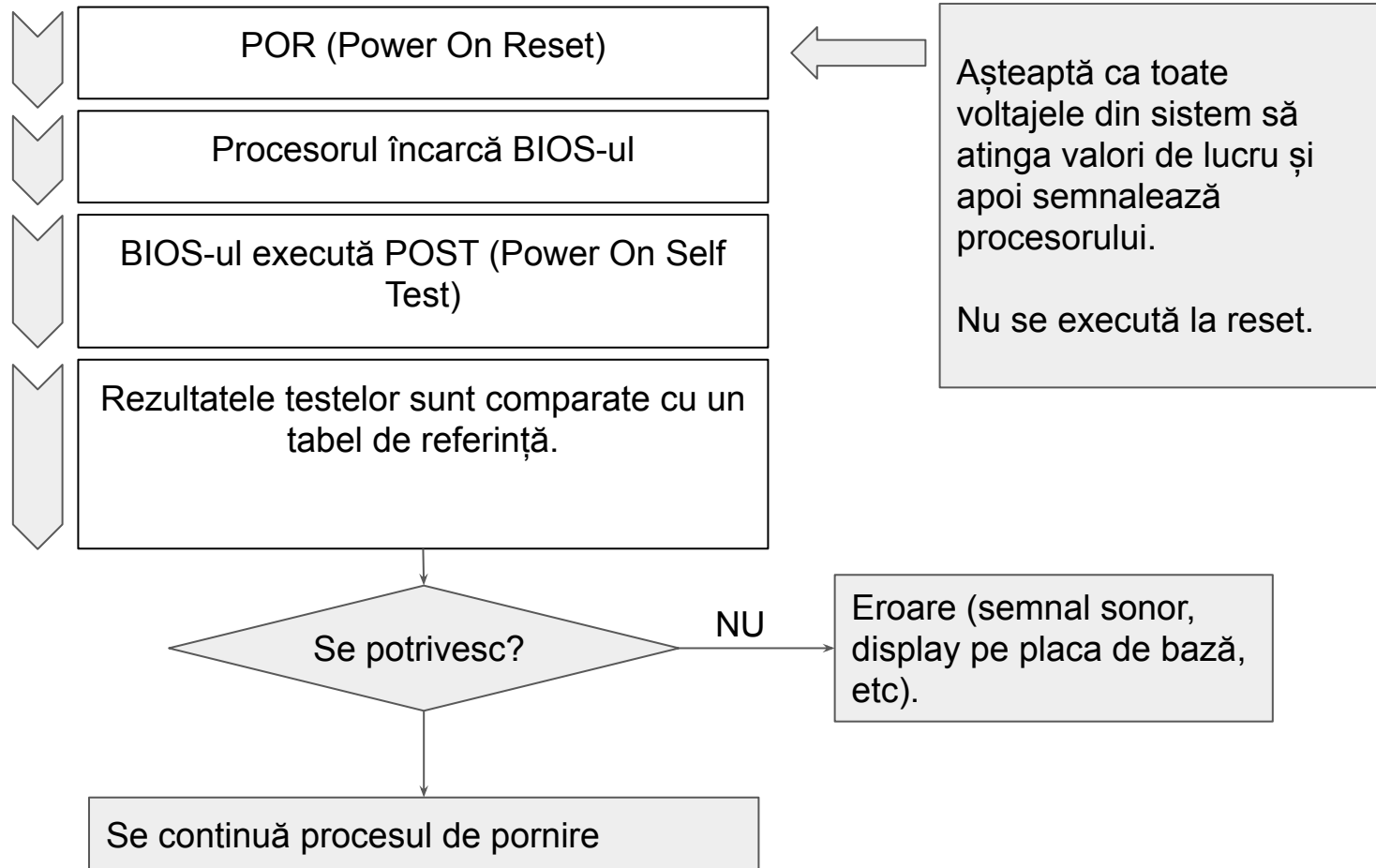
```
./scripts/dtc/fstree.c
./scripts/dtc/libfdt/fdt_empty_tree.c
./scripts/dtc/libfdt/fdt_ro.c
./scripts/dtc/libfdt/fdt_rw.c
./scripts/dtc/libfdt/fdt_strerror.c
./scripts/dtc/libfdt/fdt.c
./scripts/dtc/libfdt/fdt_wip.c
./scripts/dtc/util.c
./scripts/dtc/fdtput.c
./scripts/dtc/checks.c
./scripts/dtc/treesource.c
./scripts/dtc/flattree.c
./scripts/dtc/dtc.c
./scripts/dtc/data.c
./scripts/dtc/fdtdump.c
./scripts/dtc/fdtget.c
./scripts/dtc/srcpos.c
./scripts/dtc/livetree.c
./scripts/kallsyms.c
./scripts/unifddef.c
./scripts/genksyms/genksyms.c
./scripts/selinux/genheaders/genheaders.c
./scripts/selinux/mdp/mdp.c
./scripts/asnl_compiler.c
./scripts/pnmtologo.c
./scripts/recordmcount.c
./scripts/docproc.c
./scripts/sortextable.c
sorin@sorin: /usr/src/linux >
```

Cum pornește un sistem de operare?



BIOS	Basic Input Output System, în afara sistemului de operare
MBR	Un loc special pe harddisk în care BIOS-ul știe să caute boot loaderul
Grub	Cel mai utilizat boot loader în linux, încarcă nucleul sistemului
init/systemd	Primul proces care rulează
runlevel	Lista de programe care trebuie rulate, în funcție de modul de pornire.

BIOS (BASIC INPUT OUTPUT SYSTEM)



Procesele inițiale

Toate procesele sunt create de un alt proces, cu excepția unui proces initial:

➤ Linux: init, înlocuit cu systemd

➤ Mac OSX: launchd

➤ Windows XP: NTLDR

➤ Windows 7: NTOSkrnl



Procesele inițiale trebuie să ruleze pe toată durata rulării sistemului de operare. Orice problemă a procesului inițial duce la oprirea sistemului.

Crearea proceselor

- Crearea proceselor se face pe sistemele de tip Unix printr-un apel de sistem numit *fork*.
- Când un proces trimite acest apel de sistem, sistemul de operare va crea un nou proces care, la început, va reprezenta o copie a procesului părinte, având la dispoziție toate resursele acestuia (fișiere deschise, conexiuni de rețea deschise, etc.).
- Noul proces nu are acces la resursele de memorie și procesor ale părintelui.
- Procesul părinte poate închide toate resursele disponibile pentru noul proces, făcându-l complet independent.

Crearea proceselor

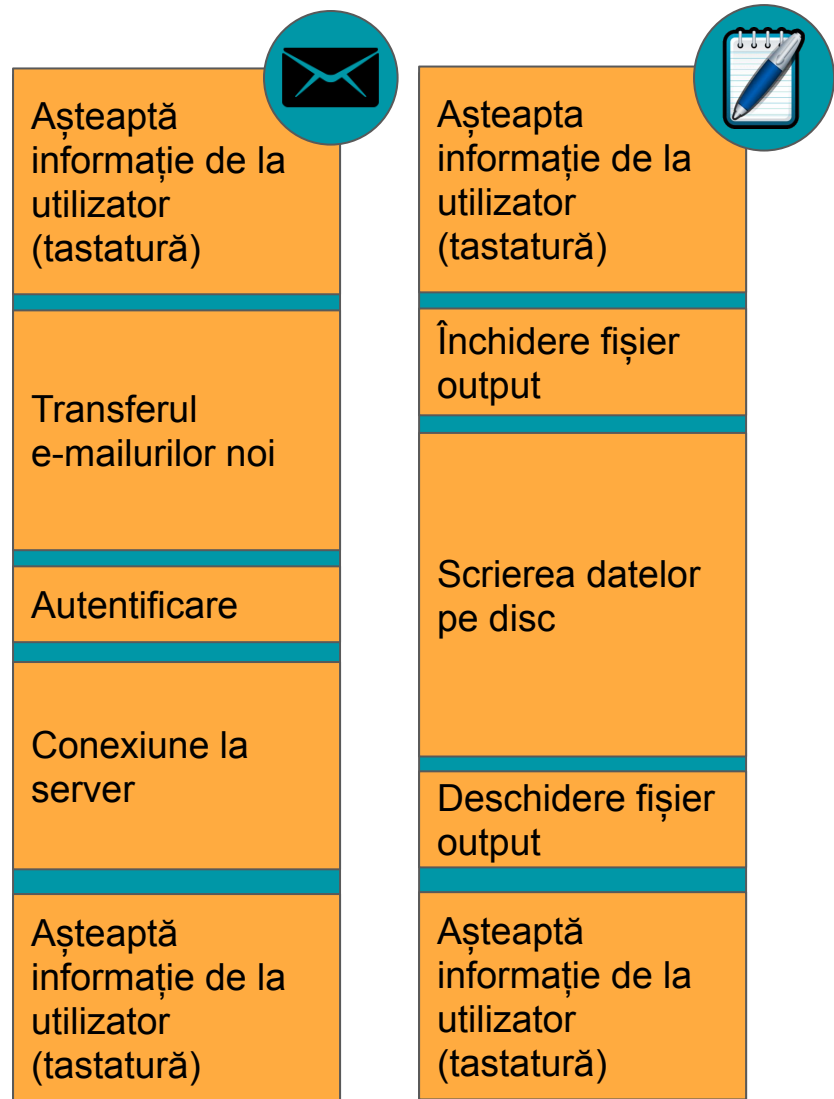
- Procesele create pot rula în paralel cu procesul părinte.
- Terminarea procesului părinte poate conduce fie la terminarea proceselor create de acesta, fie la punerea lor în starea de “proces orfan”. În Linux, aceasta se face prin procesul de “reparenting” - atribuirea procesului init ca fiind noul parinte. ie
- În Linux, procesele care rămân fără “părinte” în mod intenționat se numesc *demoni* (daemon). Acestea sunt preluate fie de init, fie de alt proces părinte și pot rula indefinit.
- Un proces de tip “zombie” este un proces care s-a încheiat dar încă apare în tabela de procese. De obicei, este vorba despre un proces al cărui părinte nu a aflat că s-a încheiat.

Crearea proceselor

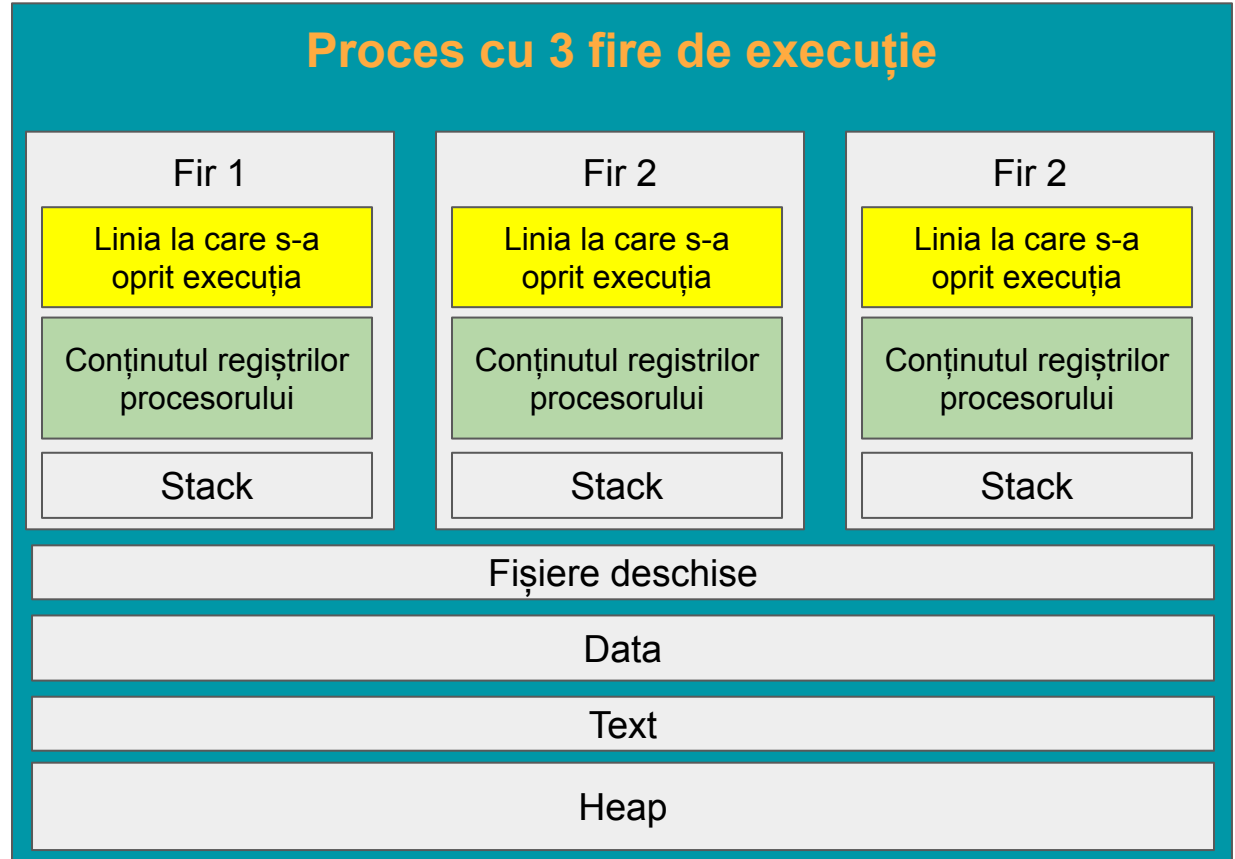
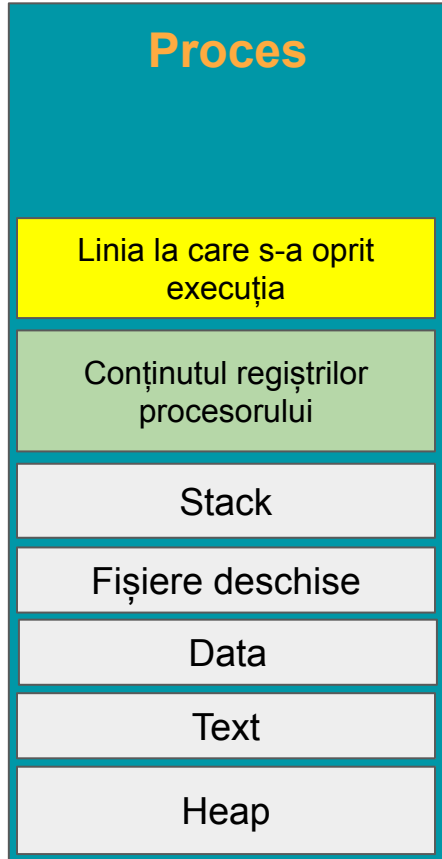
- În Linux, procesele pot lansa în execuție un alt proces în locul lor (înlocuind resursele procesului original cu cele ale noului proces).
- ID-ul procesului rămâne același dar segmentele de memorie (text, stack, heap) vor fi înlocuite cu cele ale noului proces.
- În general, lansarea în execuție a unui proces în Linux presupune un apel fork al procesului părinte, urmat de un apel exec care înlocuiește noua copie a părintelui cu procesul executat.

Fire de execuție (threads)

- Anumite programe trebuie să execute din când în când operații care durează foarte mult. Dacă un editor de text este un process, nu se va putea scrie în el în timp ce înregistrează documentul de disc, sau în timp ce îl imprima.
- O aplicație de citit e-mailuri nu va putea să permită scrierea unui e-mail în timp ce verifică existența noilor e-mailuri.
- Este posibil ca un proces să aibă două sau mai multe subprocese care să fie programate independent și să poată fi rulate independent?

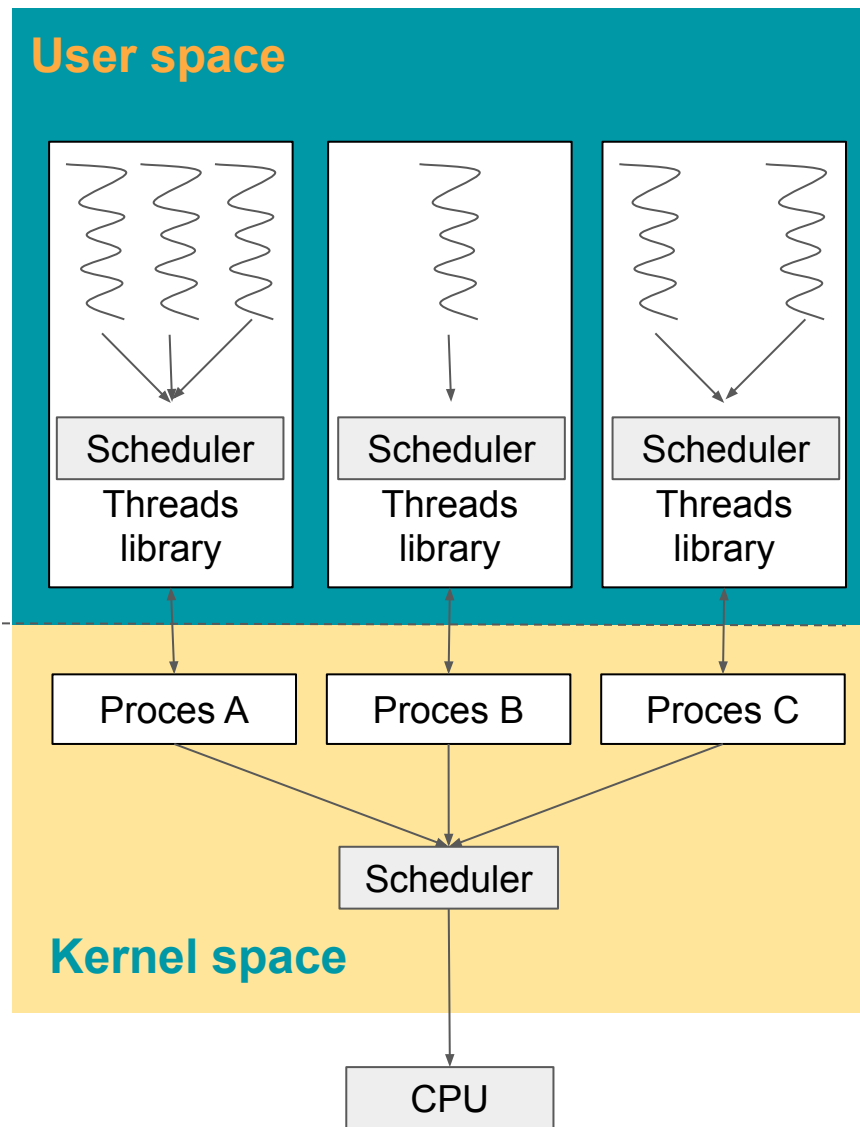


Fire de execuție și procese

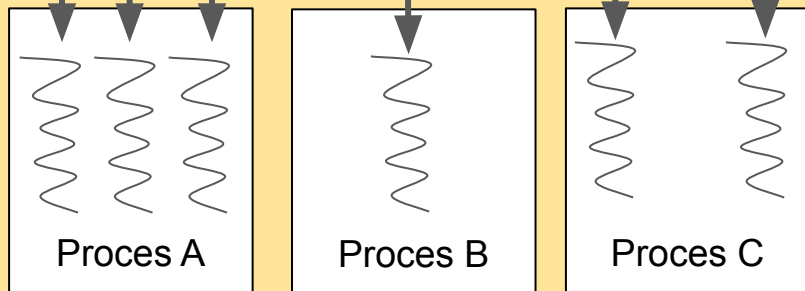
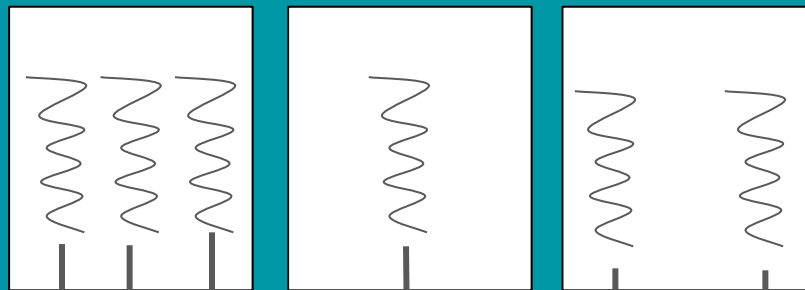


Fire de execuție în afara nucleului (green threads)

- Nucleul nu intervine în programarea firelor de execuție
- Mutarea între firele de execuție se face de către un scheduler implementat în interiorul procesului, nucleul mută între procese.
- Nu este posibil ca firele de execuție ale aceluiași proces să ruleze pe mai multe procesoare în paralel.
- Procesele care implementează astfel de fire de execuție au la bază o bibliotecă specială care se ocupă de acestea, numită "threads library" sau "runtime". Ex: POSIX pthreads, Win32 threads.
- Threadurile trebuie să cedeze controlul, nu există întreruperi hardware (dacă nu intervine schedulerul)



User space



Scheduler

CPU

Kernel space

Fire de execuție în nucleu (native threads, Lightweight processes - LWP)

- Fiecare proces este compus din unul sau mai multe fire de execuție
- Pentru fiecare fir de execuție în proces se va crea un fir de execuție în nucleu
- Mutarea se face între fire de execuție.
- Mai multe fire de execuție ale aceluiași proces pot rula pe mai multe procesoare în paralel
- Mai lente decât firele de execuție “user space” dar pot beneficia de mai mult timp de procesor și de mai multe procesoare.
- În esență, firele din nucleu fac în așa fel încât fiecare fir vede un procesor virtual.

Relația între user level threads și kernel threads

- Programatorii implementează user level threads folosind o bibliotecă specială
- Biblioteca specială decide felul în care asociază threadurile din proces cu cele din nucleu.
- E posibil ca aceeași bibliotecă (ex: JAVA) să facă asocierea diferit în funcție de sistemul de operare pe care rulează.
- Majoritatea aplicațiilor care folosesc fire de execuție în sistemele moderne, folosesc kernel threads.

Java a trecut la kernel threads în 2000

Ruby a trecut la kernel threads în 2008

Procese vs. fire de execuție: Google Chrome



Procese

Browser - procesul principal

GPU Process - accelerare hardware în placa video

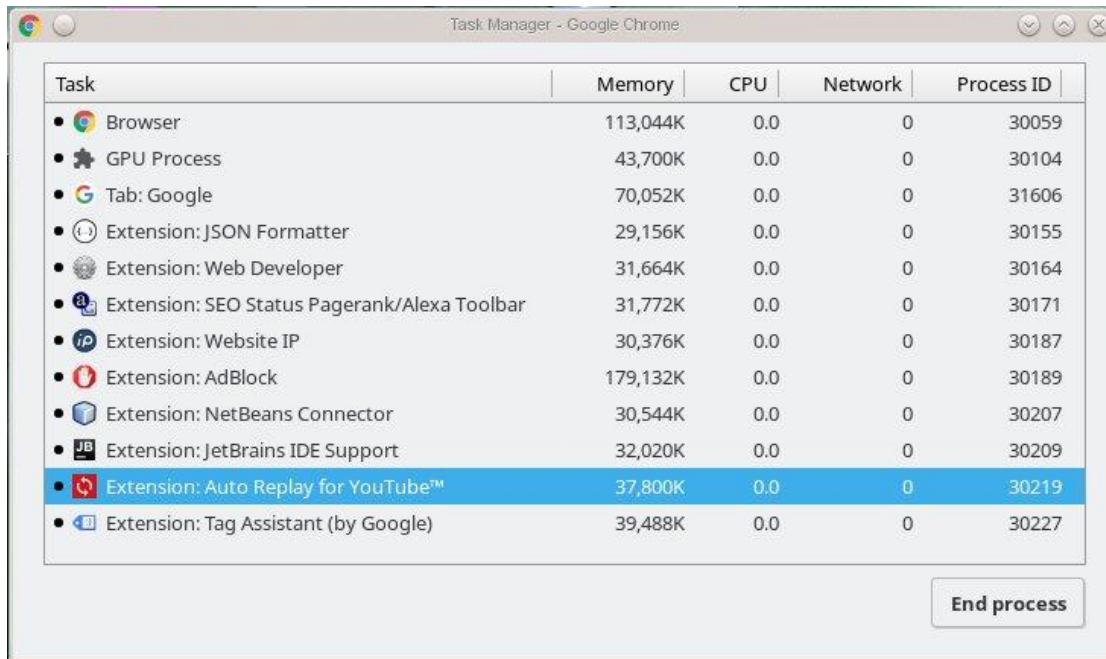
Tab: Google - Tab-ul în care este încărcată pagina google.ro

Extension: extensii ale browserului

Threaduri pentru procesul Tab:Google

31606 31606 chrome
31606 31607 TaskSchedulerSe
31606 31608 Chrome_ChildIOT
31606 31609 GpuMemoryThread
31606 31610 Compositor
31606 31611 Renderer::FILE
31606 31614 CompositorTileW
31606 31636 ScriptStreamerT
31606 31656 WorkerPool/27
31606 31657 WorkerPool/28

Chrome Task Manager



Task	Memory	CPU	Network	Process ID
• Browser	113,044K	0.0	0	30059
• GPU Process	43,700K	0.0	0	30104
• Tab: Google	70,052K	0.0	0	31606
• Extension: JSON Formatter	29,156K	0.0	0	30155
• Extension: Web Developer	31,664K	0.0	0	30164
• Extension: SEO Status Pagerank/Alexa Toolbar	31,772K	0.0	0	30171
• Extension: Website IP	30,376K	0.0	0	30187
• Extension: Adblock	179,132K	0.0	0	30189
• Extension: NetBeans Connector	30,544K	0.0	0	30207
• Extension: JetBrains IDE Support	32,020K	0.0	0	30209
• Extension: Auto Replay for YouTube™	37,800K	0.0	0	30219
• Extension: Tag Assistant (by Google)	39,488K	0.0	0	30227

End process

Procese vs. fire de execuție: Firefox



ps -T -p 2376 - 2376 id-ul procesului

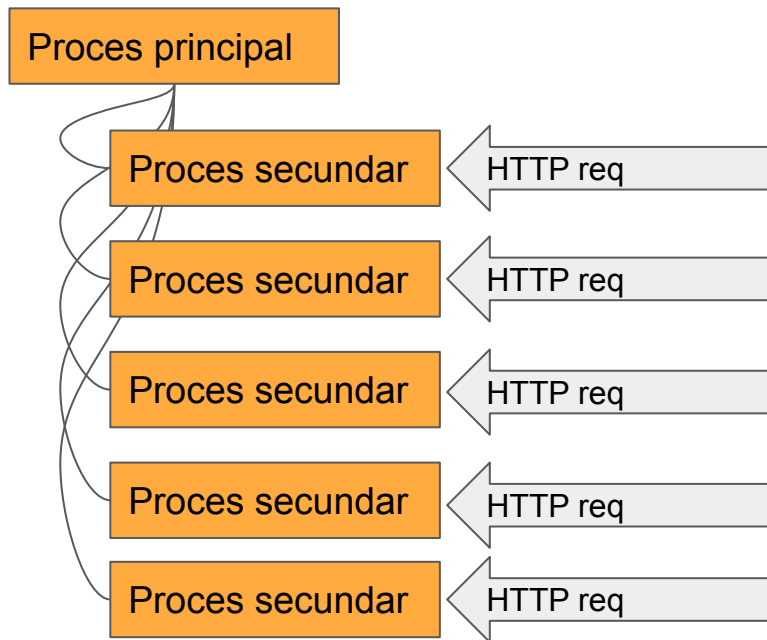
2376	2376	firefox	2376	2415	ImgDecoder #1
2376	2380	gdbus	2376	2418	ImageIO
2376	2381	Gecko_IOThread	2376	2419	SoftwareVsyncTh
2376	2382	Link Monitor	2376	2422	URL Classifier
2376	2383	Socket Thread	2376	2426	mozStorage #1
2376	2384	JS Watchdog	2376	2433	ImageBridgeChil
2376	2385	JS Helper	2376	2434	mozStorage #2
2376	2391	JS Helper	2376	2435	Proxy R~olution
2376	2392	JS Helper	2376	2436	DNS Resolver #1
2376	2393	Hang Monitor	2376	2437	mozStorage #3
2376	2402	Cache2 I/O	2376	2439	SSL Cert #1
2376	2403	Timer	2376	2443	threaded-ml
2376	2404	DataStorage	2376	2444	SSL Cert #2
2376	2406	GMPThread	2376	2445	DOM Worker
2376	2407	IPDL Background	2376	2446	localStorage DB
2376	2408	DOM Worker	2376	2451	Storage I/O
2376	2409	HTML5 Parser	2376	2460	StreamTrans #19
2376	2414	Compositor			

➤ Dacă un tab sau o fereastră a firefoxului se blochează, se blochează toate.

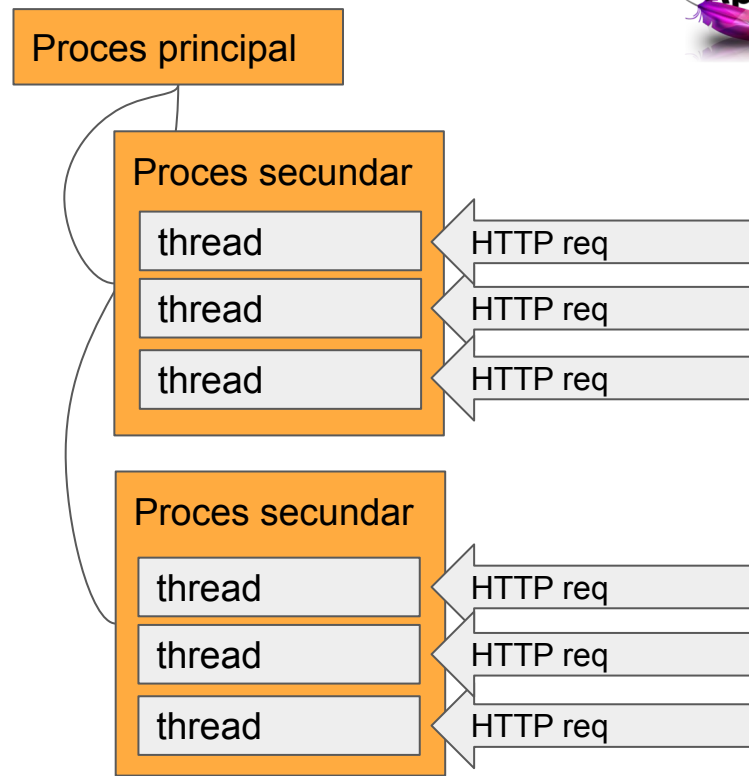
➤ Firefox a trecut in 2018 la un model mixt, odata cu aparitia Firefox Quantum.

Procese vs. fire de execuție: Apache WEB server

Prefork MPM



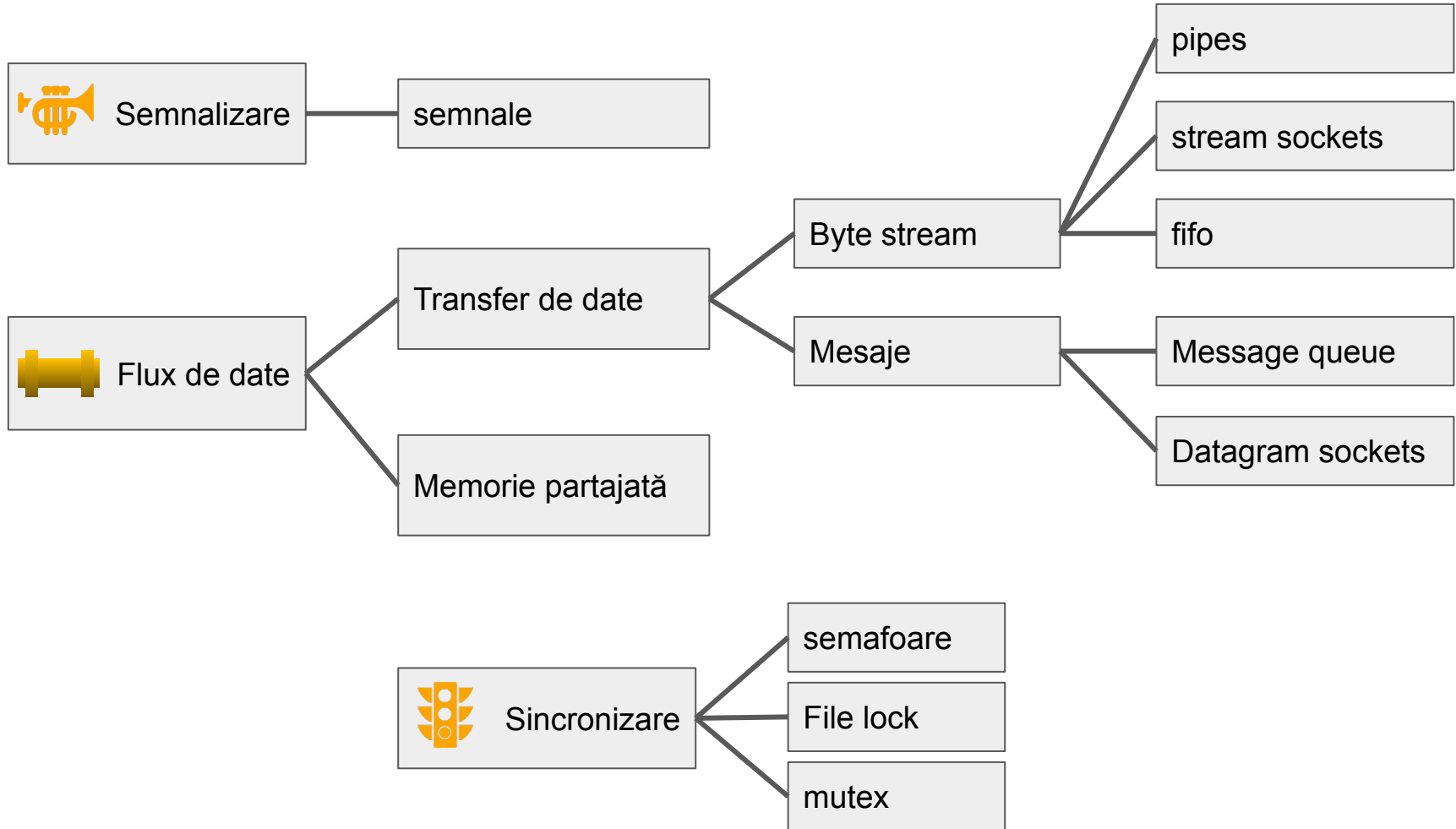
Worker MPM



Apache este multiprocesor dar procesele acestuia, în funcție de configurare, pot fi multi-thread sau nu.

Comunicarea între procese

Comunicarea între procese (IPC)



Semnale

- Semnalele sunt un sistem de notificare asincronă în care un proces informează alt proces cu privire la un anumit eveniment.
- Programele pot implementa funcții speciale care așteaptă aceste semnale și acționează în consecință sau pot decide să blocheze orice semnal, cu două excepții: SIGSTOP și SIGKILL
- Sunt disponibile 32 de semnale pe procesoarele de 32 de biți și 64 (cel mult) pe cele de 64 de biți.
- Primirea semnalelor este condiționată de permisiuni: un proces nu poate primi semnale de la un alt proces care nu este deținut de același user sau grup.
- Dacă procesul este în stare de așteptare sau blocat, semnalele nu vor fi primite până când acesta nu revine în stare de execuție.





Semnale - exemple

1 -	SIGHUP	Hangup - terminalul atașat a fost închis
2 -	SIGINT	Oprire normală a procesului (CTRL-C)
3 -	SIGQUIT	Oprire cu înregistrarea memoriei de lucru
7 -	SIGBUS	Eroare în accesarea memoriei
8 -	SIGFPE	Eroare de calcul aritmetic (ex: împărțire la 0)
9 -	SIGKILL	Oprire necondițională a procesului
10 -	SIGUSR1	Nedefinit: poate fi utilizat de programe
11 -	SIGSEGV	Eroare de segmentare
13 -	SIGPIPE	Scrierea într-un pipe a eșuat
14 -	SIGALRM	Cod de alarmă
15 -	SIGTERM	Terminare a procesului
19 -	SIGSTOP	Oprire a procesului (poate fi reluat)

Procesele pot trimite între ele aceste semnale.

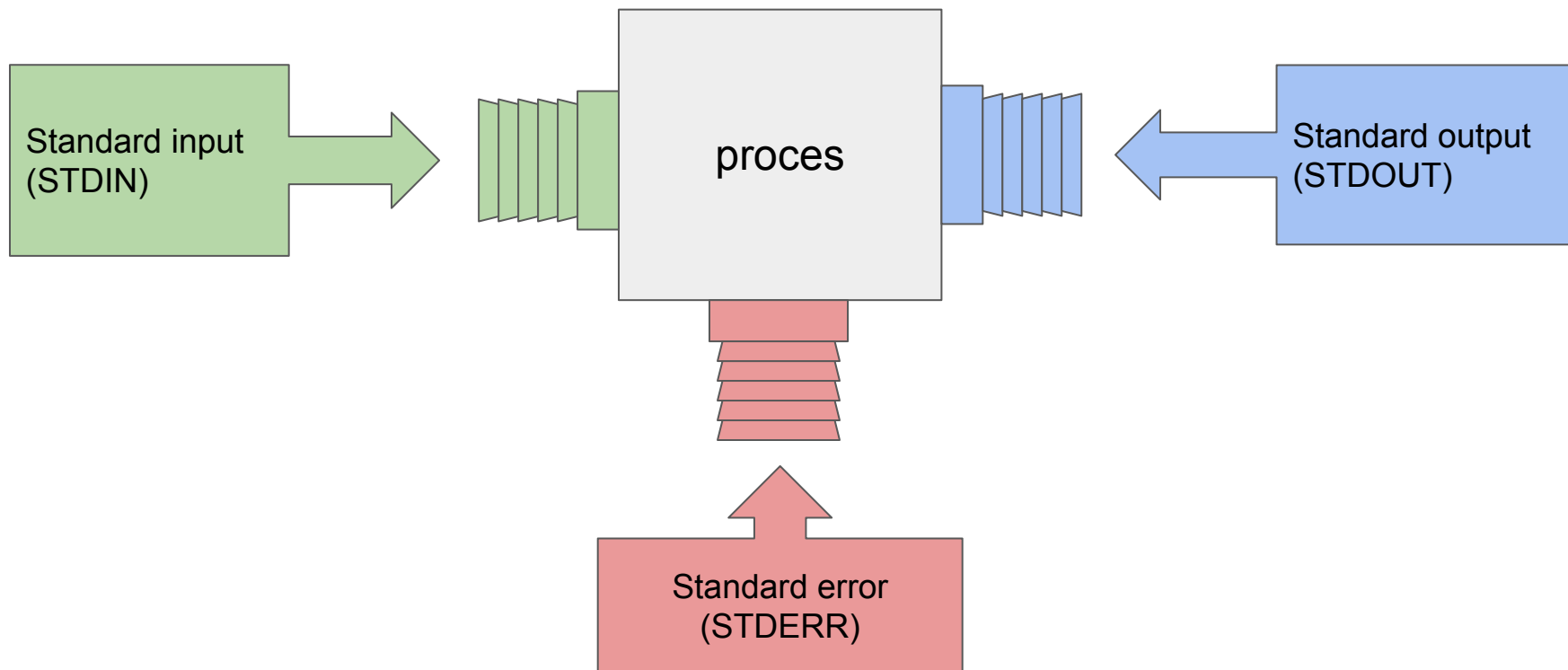
SIGHUP este folosit pentru a semnaliza programelor de tip daemon să reîncarce fișierele de configurare.

SIGKILL poate fi folosit pentru oprirea imediată a unui proces care blochează resursele sistemului.

SIGTERM este semnalul trimis de sistemul de operare către procese la oprirea computerului.

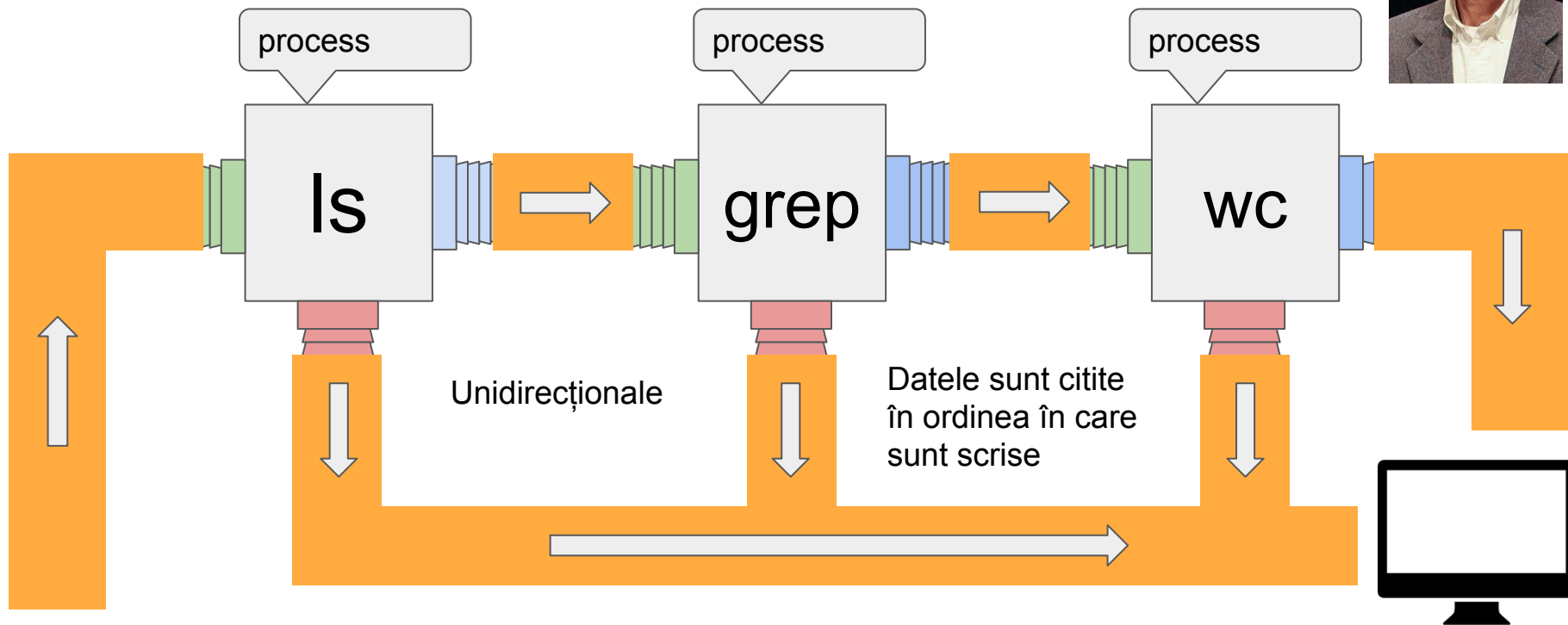
Flux de date: Conducte (pipes)

Sistemul de conduite se bazează pe existența fluxurilor standard ale unui proces. Acestea există pe toate sistemele de operare, în configurații similare.



Flux de date: Conducute (pipes)

Douglas McIlroy
Bell Labs, 1964



```
ls -R | grep '.jpg' | wc -l
```

Pipe-urile sunt inițializate de sistemul de operare la cererea proceselor. După ce procesele se termină, pipe-urile se închid.



Flux de date: Sockets (Socluri)

- Un socket este un punct final într-un sistem de comunicare (e nevoie de două)
- Bidirecțional, oferă metode de control a transferului de date
- Fiecare socket este subordonat unui “domeniu” de adrese.
- Domeniul UNIX (adresa este calea către fișier) - UNIX domain sockets
- Domeniul IPV4 (adresa este adresa IPv4 (32 biți) + numărul portului)
- Domeniul IPV6 (adresa este adresa IPv6 (128 biți) + numărul portului)

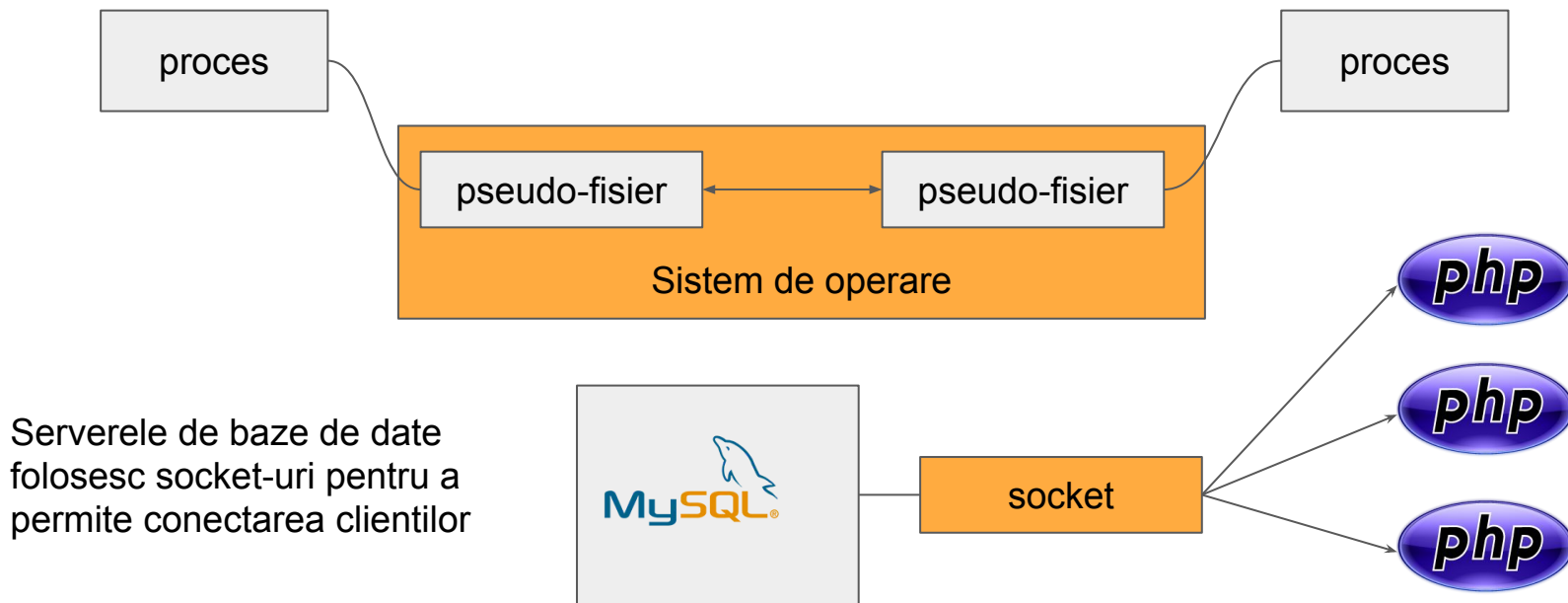
UNIX domain Sockets



Domeniul UNIX (adresa este calea către fișier) - UNIX domain sockets

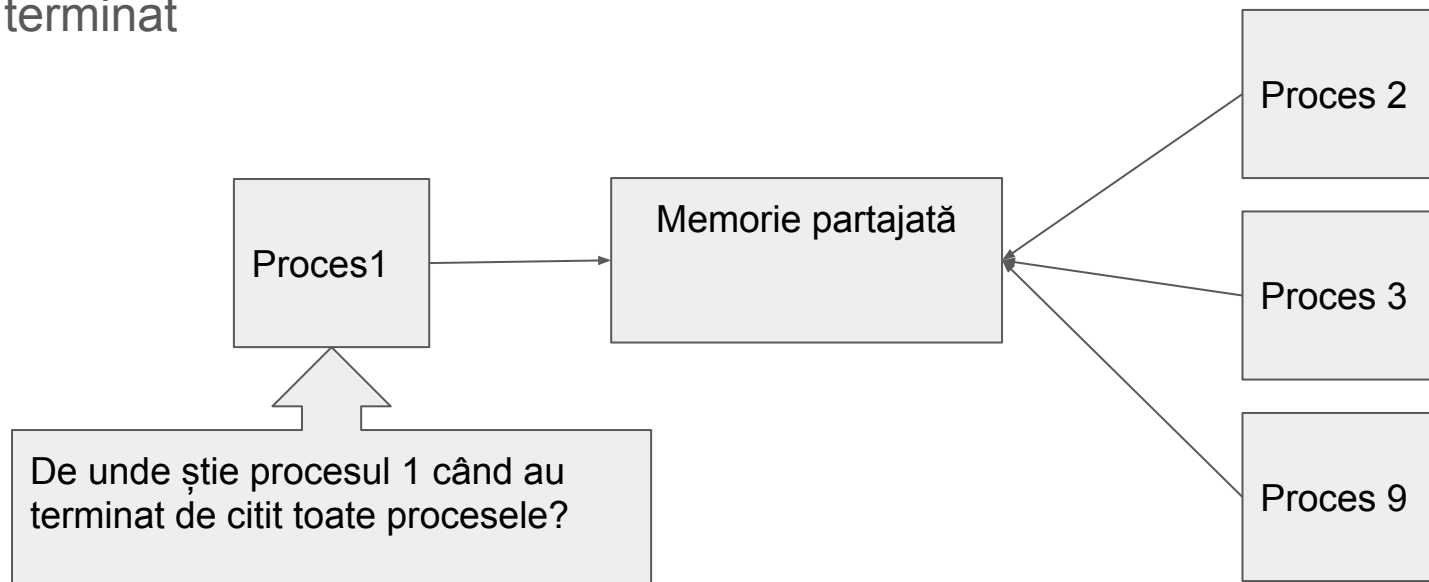
Permite atașarea mai multor procese, fiecare cu propriul său canal de date.

Disponibile pe sistemele de tip UNIX



Memorie partajată

- Datele scrise în memoria partajată ar putea fi citite de mai multe procese. Procesul 1 trebuie însă să aștepte până când toate procesele care citesc au terminat



- Accesul la zona de memorie partajată trebuie reglementat prin sincronizare. Sistemul de sincronizare se numește *sistemul de semafoare*.

DBUS și Systemd- descentralizare vs centralizare

UNIX-ul a fost prin definiție descentralizat. Fiecare aplicație făcea un singur lucru iar utilizatorul le lega între ele când avea nevoi mai complexe.

Linux-ul se deplasează spre modele din ce în ce mai centralizate și mai opace.

Avantajele centralizării

- Eficiența mult mai mare
- Control mult mai bun
- Interoperabilitate și automatizare
- Viziune comună
- Execuție rapidă
- Defecte reduse

Dezavantajele centralizării

- Greșelile se propagă foarte eficient
- Dificil de intrat în joc
- Greu de implementat operații exotice
- Greu de impus viziunea proprie
- Creativitate extrem de redusă
- Ineficiență locală.



Atracția Linuxului (și a UNIX-ului în mare măsură) a fost tocmai ușurința (relativă) cu care cineva putea să își implementeze propriile proiecte mici, fără să trebuiască să învețe librării masive.

