

# Curs 04: Planificarea execuției

## Sumar curs anterior

procese sunt încapsularea execuției într-un sistem  
procese abstractizează procesorul, memoria și I/O  
un proces este creat de un alt proces, dintr-un executabil, prin loading  
atributele unui proces sunt păstrate în PCB (Process Control Block)  
pe Unix există apelurile `fork()` și `exec()`: `fork()` duplică spațiul de adrese al unui proces, `exec()` invocă loader-ul  
procese formează o ierarhie, procesul părinte așteaptă încheierea proceselor copil  
procese orfane sunt adoptate de `init`  
procese zombie sunt cele care nu au fost încă așteptate de procesul părinte  
procese comunică pentru transfer de date și pentru notificări  
pipe-urile anonime (folosite de shell la comanda `cmd1 | cmd2`) sunt folosite doar între procese înrudite

## Multitasking și scheduling

pe un sistem există mai multe procese  
sistemul dispune de un număr limitat de procesoare  
sistemul de operare trebuie să asigure accesul tuturor proceselor la procesoare  
un proces rulează pe un procesor (sau mai multe) și apoi lasă locul altui proces:  
multi-tasking  
sistemul de operare planifică accesul unui proces la procesoare: scheduling  
planificatorul de procese este o componentă din sistemul de operare care decide când un proces părăsește procesorul și ce proces îi ia locul

## Schimbarea de context

un proces este un program căruia i se atașează un context de execuție  
când planificatorul decide că un proces părăsește procesorul, are loc o schimbare de context  
schimbarea de context înseamnă salvarea informațiilor procesului anterior într-o zonă din sistemul de operare și restaurarea informațiilor noului proces  
schimbarea de context înseamnă overhead  
schimbarea de context se poate produce:  
\* voluntar: un proces decide sau cauzează schimbarea de context  
\*\* un proces cedează de bună voie procesorul: yielding  
\*\* un proces execută o operație blocantă (de exemplu citire de la un dispozitiv care nu are încă date)  
\*\* un proces își încheie execuția  
\* nevoluntar: planificatorul sistemului de operare decide forțarea unui proces de pe procesor

- \*\* un proces a stat prea mult timp pe procesor
- \*\* apare în sistem un proces mai important
- + demo cu schimbări voluntare și nevoluntare la rularea comezii find

## Reminder: Stările proceselor

un proces care rulează pe procesor este în starea running  
un proces care execută o operație I/O se blochează în așteptarea încheierii operației, intră în starea blocking/waiting  
atunci când un proces poate rula, dar nu are alocat un procesor, este în starea ready  
stările running, blocking și ready sunt cele trei stări principale ale unui proces  
+ diagramă cu stările proceselor  
tranziția din running în ready se întâmplă când unui proces îi expiră cuanta  
tranziția din running în blocking este când un proces realizează o operație I/O blocantă  
tranziția din blocking în ready are loc când operația I/O blocantă s-a definitivat  
tranziția din ready în running este când se eliberează un procesor

## Stările proceselor și schimbările de context

tranzițiile din starea running sunt tranziții de schimbare de context: procesul cedează procesorul în fața altui proces  
când un proces cedează procesorul, se alege un proces din starea READY  
ce se întâmplă când nu există nici un proces în starea READY? procesul idle  
ce tranziții au loc la diferitele tipuri de schimbări de context:  
RUNNING -> READY: yielding, prea mult timp pe procesor, proces prioritar  
RUNNING -> WAITING: un proces execută o operație blocantă  
RUNNING -> TERMINATED: un proces și-a încheiat execuția  
noul proces execută tranziția READY -> RUNNING

## Planificatoare cooperative și preemptive

planificatorul este o componentă a sistemului de operare; este o funcție care este apelată pentru:  
\* a selecta un proces aflat în READY  
\* a înlocui procesul curent (din RUNNING) cu noul proces selectat (din READY)  
planificatorul este apelat pentru a efectua schimbarea de context  
planificatoarele pot fi cooperative și preemptive  
planificatoarele cooperative permit doar schimbări de context voluntare  
cele preemptive permit și schimbări de context nevoluntare: a preempta = a forța părăsirea procesorului  
planificarea cooperativă are dezavantajul starvation: un proces poate acapara complet procesorul și să nu lase alte procese să ruleze  
în general planificarea preemptivă presupune existența unei cuante de timp asociate unui proces; expirarea cuantei de timp forțează înlocuirea procesului (context switch)

# Criterii de evaluare a unui planificator

sunt esențiale două metrici: productivitate (throughput) și echitate (fairness)

un planificator este cu atât mai productiv cu cât se consumă cât mai puțin timp schimbând contextul și mai mult timp rulând procese; procesele să își termine cât mai repede treaba

un planificator este cu atât mai echitabil cu cât fiecare proces are acces la procesor; adică procesele stau cât mai puțin timp în coada READY până să intre în RUNNING

un sistem este inechitabil când un proces stă foarte mult în coada READY și nu este planificat pe procesor: starvation

un sistem este neproductiv când sunt schimbări de context foarte dese și procesoarele fac foarte puțină treabă

turnaround time: timpul din care un proces intră în sistem până când își încheie execuția

average turnaround time: media turnaround time pentru toate procesele din sistem

waiting time: suma timpilor în care un proces așteaptă în starea READY

average waiting time: media waiting time pentru toate procesele din sistem

un sistem productiv are average turnaround time mic

un sistem echitabil are average waiting time mic

Atenție: waiting time se referă la timpul petrecut în starea READY (nu în starea WAITING)

+ demo cu timpul de așteptare (în coada READY)

## Cuanta de timp. Planificatorul round robin

cuanta de timp se asociază unui proces în planificatoare preemptive

când expiră cuanta unui proces acesta este scos de pe procesor, i se calculează o nouă cuantă și este trecut în starea READY

timer-ul procesorului (uzual o dată 1 ms sau 10ms) generează întreruperi de ceas; în rutina de tratare a întreruperii de ceas se verifică dacă un proces în RUNNING are cuanta expirată  
cuanta este dinamică, se actualizează la fiecare expirare

productivitate vs echitate: cuantă mare vs cuantă mică

cuantă mare -> schimbări de context mai rare, productivitate sporită

cuantă mică -> schimbări de context mai dese, echitate sporită

cuanta variază între procese; procesele sunt CPU-intensive vs I/O-intensive

procese I/O-intensive primesc în general o cuantă de timp mai mare pentru că se vor bloca și vor trece din RUNNING în WAITING și vor lăsa alt proces în loc

procese CPU-intensive primesc în general o cuantă de timp mai mică; este posibil să ruleze pe procesor până la expirarea cuantei; cu o cuantă mare ar dura mai mult până ar lăsa alt proces în locul său pe procesor

## Coadă/cozi de procese READY. Prioritățile proceselor

când are loc o schimbare de context, planificatorul alege un proces aflat în starea READY și îl trece în RUNNING; ce procese alege

planificatorul round-robin are o coadă de procese READY; ia pe rând procesele READY și le trece în RUNNING; când un proces ajunge în READY e adăugat la sfârșitul cozii

unele procese sunt mai importante; au prioritate mai mare

planificatoarele cu priorități au mai multe cozi pentru procesele READY: câte o coadă pe o prioritate; când se planifică un proces se ia primul proces din coada cu prioritatea cea mai mare

prioritate statică este o prioritate care nu poate fi modificată pe parcursul execuției procesului; un sistem care folosește doar priorități statice poate duce la starvation pentru procesele cu prioritate mai mică; vor rula mereu procesele cu prioritate mai mare dacă dintre acestea sunt mereu în starea READY

prioritățile statice în Linux sunt numite "nice"

+ demo cu procese cu valoare nice diferită

pentru a preveni starvation, prioritățile sunt dinamice, se modifică pe parcursul execuției; un proces care stă destul de mult într-o coadă READY cu prioritate mai mică va fi promovat într-o coadă READY cu prioritate mai mare

modificarea priorității ține cont de natura procesului, similar alegerii cuantei: procesele CPU-intensive primesc, în general, o prioritate mai mică decât procesele I/O intensive

## Comunicarea inter-proces

procesele comunică pentru a transfera informații, pentru notificare sau pentru a asigura integritatea datelor (sincronizare)

transferul de informații se face prin

\* comunicare de tip țeavă, sau transfer de mesaje: pipe-uri (anonime), pipe-uri cu nume (FIFO), socketi locali (UNIX), socketi de rețea (Berkeley), cozi de mesaje

\*\* există API-uri și biblioteci de message passing: MPI, ZeroMQ, RabbitMQ

\* comunicare cu date partajate: memorie partajată (shared memory)

\*\* API de date partajate: OpenMP

la comunicarea prin transfer de mesaje există un sender, un receiver, un canal virtual de comunicare și un protocol înțeles de parteneri; în general nu e nevoie de sincronizarea accesului la date pentru că fiecare partener are o instanță

la comunicarea prin memorie partajată partenerii pot fi cititori sau scriitori; datele fiind comune, este nevoie de sincronizare pentru a asigura integritatea acestora

notificarea se face prin semnale (numite și excepții pe Windows); un proces își definește o rutină de tratare a semnalului/excepției, rutină apelată la apariția semnalului

semnalele pot fi livrate de

\* sistemul de operare ca urmare a întâmpinării unei situații neprevăzute în execuția procesului: de exemplu acces nevalid la memorie (segmentation fault)

\* de un alt proces pentru a notifica procesul curent de o condiție sau eveniment

sincronizarea se face prin mutex-uri, semafoare, variabile condiție; mai multe au fost prezentate la APD, vom insista la cursul 9: Sincronizare

# Sumar

mai multe procese concurează pe procesoarele sistemului; planificatorul (parte a sistemului de operare) gestionează accesul proceselor la procesoare (trecerea în starea RUNNING) planificarea unui proces, adică înlocuirea unui proces cu altul se numește schimbare de context

schimbarea de context se poate produce:

- \* voluntar: un proces decide sau cauzează schimbarea de context

- \*\* un proces cedează de bună voie procesorul: yielding

- \*\* un proces execută o operație blocantă (de exemplu citire de la un dispozitiv care nu are încă date)

- \*\* un proces își încheie execuția

- \* nevoluntar: planificatorul sistemului de operare decide forțarea unui proces de pe procesor

- \*\* un proces a stat prea mult timp pe procesor

- \*\* apare în sistem un proces mai important

planificatoarele pot fi cooperative și preemptive; cele cooperative nu pot preveni apariția situației de starvation

planificatorul urmărește două obiective conflictuale: productivitate și echitate

planificatoarele preemptive folosesc cuantă de timp și prioritate pentru fiecare proces

comunicarea între procese presupune transfer de mesaje, memorie partajată, notificare sau sincronizare