

Capitol 03: Procese

Sumar curs anterior

stocăm datele în fișiere, fișierele sunt date agregate/compartimentate, metadatele fișierelor sunt agregate în FCB (file control block)

Fișierele sunt deschise pentru a fi folosite, se întoarce la nivel low-level un descriptor de fișier

un descriptor este o intrare într-o tabelă de descriptori de fișiere care conține un pointer către o structură de fișier deschis

mai multe structuri de fișier deschis pot referi același FCB (mai multe apeluri open(2) pe același fișier)

mai multe intrări în tabela de descriptori de fișiere pot referi aceeași structură de fișier deschis (folosind dup(2) sau dup2(2))

operațiile read(2), write(2) și lseek(2) modifică cursorul de fișier (prezent în structura de fișier deschis)

operația ftruncate(2) modifică dimensiunea fișierului (prezentă în FCB)

Acțiuni în sistemul de calcul

utilizatorul dorește execuția de acțiuni în sistem: folosirea procesorului/procesoarelor datele sunt în memorie (aduse acolo de la I/O, eventual suport persistent) și apoi aduse pe procesor

în memorie avem date și cod (instrucțiuni)

+ diagramă cu procesor, memorie, date

dorim să executăm mai multe acțiuni diferite pe un sistem: folosim procese

Procese

încapsularea unei acțiuni în sistemul de calcul: date și cod în memorie, rulare instrucțiuni pe procesor, interacțiune cu I/O

+ diagramă cu proces care abstractizează: memorie, procesor, I/O

permite multi-programare: mai multe acțiuni pe sistem

procesele sunt izolate între ele: memoria este separată, rulează separat pe procesor, folosesc secvențial I/O

sistemul de operare se ocupă de izolarea și planificarea proceselor

nevoia de procese: mai multe acțiuni, procesoare multiple

provocări legate de procese: izolare, planificarea accesului la resurse, comunicarea inter-procese, accesul mai multor procese la resurse limitate

Atributele unui proces

identificator (PID)

resurse: spațiu virtual de adrese (memorie), timp de lucru pe procesor, fișiere deschise (în tabela de descriptori de fișiere)

user/group

starea unui proces (mai târziu în curs)

+ demo cu `/proc/pid/status`

+ `ps -eF` (afișează atribute)

structura de proces se numește PCB (process control block), reținută în kernel space

+ demo: afișarea structurii `task_struct` din Linux:

<https://elixir.bootlin.com/linux/v4.20.13/source/include/linux/sched.h#L590>

Crearea unui proces

dintr-un proces existent (procesul părinte), uzual un shell

se creează o ierarhie de procese: un proces are un singur proces părinte dar oricâte procese copil

+ demo cu vizualizarea ierarhiei de procese

PID-ul procesului părinte este determinat cu apelul `getppid()`; PID-ul proceselor copil e întors de apelul de creare

procesul părinte invocă loader-ul care încarcă datele și codul dintr-un executabil în memoria noului proces: loading, load-time

executabilul/programul este imaginea procesului: datele (variabile globale inițializate `.data`, neinițializate `.bss` și read-only `.rodata`) și codul (`.code` sau `.text`)

executabilul are un punct de intrare (entry point) de unde începe execuția noului proces (prin acela se ajunge la funcția `main()`)

+ diagramă cu shell-ul, comanda `ls`, executabilul `/bin/ls` și crearea unui nou proces

procesele sunt identificate prin PID, nu prin nume, mai multe procese pot avea aceeași imagine de executabil

`fork()` și `exec()`

În Linux, crearea unui proces nou se face cu două apeluri: `fork()` și `exec()`

apelul `fork()` creează un proces copil ca fiind o copie a procesului părinte; partajează informații precum tabela de descriptori de fișiere, pornesc de la același cod

+ demo cu partajarea cursorului de fișier

apelul `fork()` se apelează o dată și se întoarce de două ori: o dată în procesul copil și o dată în procesul părinte

+ exemplu apel `fork()`

apoi apelul `exec()` invocă loaderul și încarcă o nouă imagine de executabil

apelul `exec()` modifică spațiul virtual de adrese al procesului, fără a schimba PID-ul acestuia

Încheierea unui proces

normală și anormală

normală: se ajunge la sfârșitul codului sau apelează `exit()`

anormală: procesul execută o acțiune nevalidă și este omorât de sistemul de operare (i se trimite un semnal) sau procesul este omorât de un alt proces (tot printr-un semnal)

rolul procesului părinte este de a se îngriji de colectarea de informații legate de încheierea procesului copil

spunem că procesul părinte așteaptă (`wait`) după procesul copil; așteptarea duce la furnizarea de informații despre modul în care și-a încheiat execuția

atunci când shell-ul creează un proces de obicei așteaptă încheierea sa (apelează `wait()`)

dacă rulăm o comandă cu `&` la sfârșit (pentru rulare în background), shell-ul nu așteaptă încheierea comenzii

un proces orfan este un proces al cărui proces părinte s-a încheiat; procesele orfane sunt înfiatate în general de procesul `init`

un proces zombie este un proces care și-a încheiat execuția dar nu a fost așteptat de părintele său

dacă un proces părinte moare și nu a așteptat un proces copil zombie, acesta este zombie orfan; este înfiat de procesul `init` și apoi este "curățat" din sistem

+ demo cu procese orfane și zombie

Procese daemon

sunt procese detașate de terminal, `stdin`, `stdout`, `stderr` nu referă terminale, de obicei `/dev/null`

procesul părinte este `init`

nu sunt interactive, realizează acțiuni de mentenanță sau oferă servicii

Rularea proceselor

procesele sunt planificate să ruleze pe procesoare

în mod normal fiecare proces are asociată o cuantă de rulare, când expiră, sistemul de operare plasează alt proces pe procesor

schimbarea unui proces cu un alt proces poartă numele de schimbare de context

mai multe detalii în cursul 4: Planificarea execuției

unele procese sunt CPU-intensive dacă petrec mult timp pe procesor sau I/O intensive dacă operează adesea pe dispozitive I/O

+ demo cu CPU intensive și I/O intensive

Starea 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

mai multe în cursul 4: Planificarea execuției

Comunicarea între procese. Pipe-uri

comunicarea inter-proces (IPC: Inter-Process Communication)

transfer de informații, notificări

apelul wait() e o formă de IPC de notificare

socketii sunt o formă de transfer de informații

pipe-uri: cu nume (FIFO, intrare în sistemul de fișiere) și anonime (doar în memorie)

+ demo cu pipe-uri cu nume

pipe-uri anonime (numite simplu pipe-uri): folosite în shell la comanda `cmd1 | cmd2`

un pipe este un buffer în kernel cu două capete: unul de citire și unul de scriere

pot fi folosite doar între procese înrudite

se redirectează stdout-ul unui proces la `pipefd[1]` (capătul de scriere)

se redirectează stdin-ul celui alt proces la `pipefd[0]` (capătul de citire)

+ demo cu pipe-uri anonime cu proces părinte, proces copil

Sumar

procesele sunt încapsularea execuției într-un sistem

procesele 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

procesele formează o ierarhie, procesul părinte așteaptă încheierea proceselor copil

procesele orfane sunt adoptate de `init`

procesele zombie sunt cele care nu au fost încă așteptate de procesul părinte

procesele comunică pentru transfer de date și pentru notificări

pipe-urile anonime (folosite de shell la comanda `cmd1 | cmd2`) sunt folosite doar între procese înrudite