Curs 03: Procese

Sumar curs anterior

stocăm datele în fişiere, fişierele sunt date agregate/compartimentate, metadatele fişierelor sunt agregate în FCB (file control block)
Fişierele sunt deschise pentru a fi folosite, se întoarce la nivel low-level un descriptor de fişier
un descriptor este o intrare într-o tabelă de descriptori de fişiere care conţine un pointer către o structură de fişier deschis
mai multe structuri de fişier deschis pot referi acelaşi FCB (mai multe apeluri open(2) pe acelaşi fişier)
mai multe intrări în tabela de descriptori de fişiere pot referi aceeaşi structură de fişier deschis (folosind dup(2) sau dup2(2))
operaţiile read(2), write(2) şi lseek(2) modifică cursorul de fişier (prezent în structura de fişier deschis)
operaţia ftruncate(2) modifică dimensiunea fişierului (prezentă în FCB)

Acţiuni în sistemul de calcul

utilizatorul doreşte execuţia de acţiuni în sistem: folosirea procesorului/procesoarelor
datele sunt în memorie (aduse acolo de la I/O, eventual suport persistent) şi apoi aduse pe procesor
în memorie avem date şi cod (instrucţiuni)
+ diagramă cu procesor, memorie, date
dorim să executăm mai multe acţiuni diferite pe un sistem: folosim procese

Procese

încapsularea unei acţiuni în sistemul de calcul: date şi cod în memorie, rulare instrucţiuni pe procesor, interacţiune cu I/O
+ diagramă cu proces care abstractizează: memorie, procesor, I/O
permite multi-programare: mai multe acţiuni pe sistem
procesele sunt izolate între ele: memoria este separată, rulează separat pe procesor, folosesc secvenţial I/O
sistemul de operare se ocupă de izolarea şi planificarea proceselor
nevoia de procese: mai multe acţiuni, 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: spaţiu virtual de adrese (memorie), timp de lucru pe procesor, fişiere deschise (în tabela de descriptori de fişiere)
user/group
starea unui proces (mai târziu în curs)
+ demo cu /proc/pid/status
+ ps -eF (afişează atribute)
structura de proces se numeşte PCB (process control block), reţinută în kernel space
+ demo: afişarea 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 şi codul dintr-un executabil în memoria noului proces: loading, load-time
executabilul/programul este imaginea procesului: datele (variabile globale iniţializate .data, neiniţializate .bss şi read-only .rodata) şi codul (.code sau .text)
executabilul are un punct de intrare (entry point) de unde începe execuţia noului proces (prin acela se ajunge la funcţia main())
+ diagramă cu shell-ul, comanda ls, executabilul /bin/ls şi crearea unui nou proces
procesele sunt identificate prin PID, nu prin nume, mai multe procese pot avea aceeaşi imagine de executabil

fork() şi exec()

în Linux, crearea unui proces nou se face cu două apeluri: fork() şi exec()
apelul fork() creează un proces copil ca fiind o copie a procesului părinte; partajează informaţii precum tabela de descriptori de fişiere, pornesc de la acelaşi cod
+ demo cu partajarea cursorului de fişier
apelul fork() se apelează o dată şi se întoarce de două ori: o dată în procesul copil şi o dată în procesul părinte
+ exemplu apel fork()
apoi apelul exec() invocă loaderul şi încarcă o nouă imagine de executabil
apelul exec() modifică spaţiul virtual de adrese al procesului, fără a schimba PID-ul acestuia

Încheierea unui proces

normală şi anormală
normală: se ajunge la sfârşitul codului sau apelează exit()

anormală: procesul execută o acţiune nevalidă şi 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 informaţii legate de încheierea procesului copil
spunem că procesul părinte aşteaptă (wait) după procesul copil; aşteptarea duce la furnizarea de informaţii despre modul în care şi-a încheiat execuţia
atunci când shell-ul creează un proces de obicei aşteaptă încheierea sa (apelează wait())
dacă rulăm o comandă cu & la sfârşit (pentru rulare în background), shell-ul nu aşteaptă încheierea comenzii
un proces orfan este un proces al cărui proces părinte s-a încheiat; procesele orfane sunt înfiate în general de procesul init
un proces zombie este un proces care şi-a încheiat execuţia dar nu a fost aşteptat de părintele său
dacă un proces părinte moare şi nu a aşteptat un proces copil zombie, acesta este zombie orfan; este înfiat de procesul init şi apoi este "curăţat" din sistem
+ demo cu procese orfane şi zombie

Procese daemon

sunt procese detaşate de terminal, stdin, stdout, stderr nu referă terminale, de obicei /dev/null
procesul părinte este init
nu sunt interactive, realizează acţiuni de mentenanţă 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 execuţiei
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 şi I/O intensive

Starea proceselor

un proces care rulează pe procesor este în starea running
un proces care execută o operaţie I/O se blochează în aşteptarea încheierii operaţiei, intră în starea blocking/waiting
atunci când un proces poate rula, dar nu are alocat un procesor, este în starea ready
stările running, blocking şi ready sunt cele trei stări principale ale unui proces
+ diagramă cu stările proceselor
tranziţia din running în ready se întâmplă când unui proces îi expiră cuanta

tranziţia din running în blocking este când un proces realizează o operaţie I/O blocantă
tranziţia din blocking în ready are loc când operaţia I/O blocantă s-a definitivat
tranziţia din ready în running este când se eliberează un procesor
mai multe în cursul 4: Planificarea execuţiei

Comunicarea între procese. Pipe-uri

comunicarea inter-proces (IPC: Inter-Process Communication)
transfer de informaţii, notificări
apelul wait() e o forma de IPC de notificare
socketii sunt o formă de transfer de informaţii
pipe-uri: cu nume (FIFO, intrare în sistemul de fişiere) şi 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 capate: unul de citire şi 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 celuilalt proces la pipefd[0] (capătul de citire)
+ demo cu pipe-uri anonime cu proces părinte, proces copil

Sumar

procesele sunt încapsularea execuţiei într-un sistem
procesele abstractizează procesorul, memoria şi 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() şi exec(): fork() duplică spaţiul de adrese al unui proces, exec() invocă loader-ul
procesele formează o ierarhie, procesul părinte aşteaptă încheierea proceselor copil
procesele orfane sunt adoptate de init
procesele zombie sunt cele care nu au fost încă aşteptate de procesul părinte
procesele comunică pentru transfer de date şi pentru notificări
pipe-urile anonime (folosite de shell la comanda cmd1 | cmd2) sunt folosite doar între procese înrudite