# Sisteme de operare

Laborator 9

Semestrul I 2024-2025

# Laborator 9

- comunicare inter-procese (IPC)
- System V IPC
  - semafoare
  - memorie partajata

# System V IPC

- cozi de mesaje, semafoare, memorie partajata
- toate aceste mecanisme folosesc un set comun de functii
- in kernel, toate structurile de date corespunzatoare sunt identificate printr-un ID (intreg nenegativ)
- pentru a folosi aceste mecanisme IPC e necesar sa stim acest ID
  - de ex: ca sa trimitem/primim mesaje intr-o/dintr-o coada de mesaje avem nevoie de ID-ul cozii
- in schimb, apelurile de creare a acestor mecanisme IPC necesita o cheie (o valoare de tip *key_t*) care va fi convertita de kernel intr-un ID

# IPC rendez-vous

- pt. ca doua procese sa comunice prin mecanisme System V IPC trebuie sa poata identifica aceeasi structura IPC din kernel
- in acest sens, exista 3 posibilitati

1) IPC_PRIVATE
   - un proces (*server*) creeaza o noua structura IPC folosind o cheie de tip IPC_PRIVATE
   - stocheaza ID-ul returnat de kernel undeva accesibil altui proces (*client*) care participa la IPC

ex: in fisier sau partajat prin *fork*

2) serverul si clientul specifica o cheie intr-un header comun

Pb: daca exista in kernel o alta structura care foloseste aceeasi cheie, creearea noii structuri IPC esueaza

# IPC rendez-vous (cont.)

3) serverul si clientul folosesc o cale comuna catre un fisier si un ID de proiect
   - le convertesc intr-o cheie cu ajutorul functiei *ftok*

- pentru a accesa o structura IPC existenta se foloseste cheia cu care a fost creata

- creare structura IPC noua
  - pt a evita folosirea unei structuri existente, apelurile de creeare a mecanismelor IPC folosesc IPC_CREAT | IPC_EXCL

- permisiuni structura IPC din kernel:
  - UID/GID proprietar
  - UID/GID creator
  - mod de acces
  - nr de secventa
  - cheie

# Structuri IPC, pros & cons

- system-wide, fara reference count
  - ex: coada de mesaje cu mesaje in ea nu este stearsa din sistem cand procesul a terminat executia
  - e nevoie de apeluri sistem explicite (sau comenzi shell) pentru a o sterge

  Ex: *$ ipcs*

  *$ ipcrm –{m|s|q} <id>*
  - prin comparatie, *pipe*-urile dispar odata cu procesele care le utilizeaza
    - FIFO: ramane numele de fisier, dar datele sunt sterse
- necunoscute sistemului de fisiere
  - e nevoie de comenzi si apeluri sistem speciale pentru a lucra cu ele
- DAR, sunt reliable, au flow control, orientate pe inregistrari, pot fi procesate si in alta ordine decat FIFO

# Semafoare System V

- complica paradigma generala
  - lucreaza cu set-uri de semafoare
  - crearea unui semafor (set, de fapt) e independenta de initializarea sa
    - consecinta: nu putem crea si initializa atomic un semafor
  - ca orice structura de date System V IPC continua sa existe in kernel si dupa ce nu mai exista procese care sa o utilizeze
    - daca un program termina fara sa elibereze semafoarele pe care le-a alocat, avem nevoie de o operatie de *undo*
- un semafor este reprezentat de o structura cu urmatoarele campuri
  - *semval* – valoarea semaforului, > 0
  - *sempid* – PID-ul procesului care a efectuat ultima operatie asupra semaforului
  - *semncnt* – nr proceselor care asteapta ca semval > valoarea curenta
  - *semzcnt* – nr proceselor care asteapta ca semval = 0

# Creare & intializare semafoare

- creare

*int semget(key_t key, int nsems, int flag);*

  - *key* se obtine cu IPC_PRIVATE sau *ftok*
  - *nsems* = nr semafoarelor din set (nsems = 0, acces la un set de semafoare existent)
  - intoarce un *semid*

- initializare

*int semctl(int semid, int semnum, int cmd, union semun arg);*

```
union semun {
        int val;                    // setare valoare semafor
        struct semid_ds *buf;       // acces/setare structura de date semafor
        ushort  *array;             // GETALL si SETALL pe setul de semafoare
}
```

# Comenzi semctl

- IPC_STAT – citire structura de date semafor
- IPC_SET – setare permisiuni (UID/GID, mode)
- IPC_RMID – sterge semaforul din sistem imediat, alte procese care incearca operatii asupra semaforului primesc EIDRM
- GETVAL – intoarce valoarea *semval*
- SETVAL – seteaza valoarea *semval* la *arg.val*
- GETALL – citeste toate valorile *semval* din setul de semafoare in *arg.array*
- SETALL – seteaza toate valorile *semval* din setul se semafoare la valorile din *arg.array*
- samd

# Operatii cu semafoare

- folosesc o structura de date de tipul

```
struct sembuf {
        ushort   sem_num;        // nr semafor in set: 0, 1 , …
        short    sem_op;         // operatia > 0, 0, < 0
        short    sem_flg;        // IPC_NOWAIT, SEM_UNDO
}
```

*int semop(int semid, struct sembuf semoparray[], size_t nops);*

- executa atomic cele *nops* operatii din *semoparray*

# Semnificatia sem_op

- valoare pozitiva (eliberare resurse)
  - valoarea lui *sem_op* este adaugata la *semval*
  - pt SEM_UNDO, *sem_op* se scade din valoarea de ajustare a semaforului pt acest proces
- valoare negativa (alocare resurse)
  - *semval* >= *abs(sem_op)* => se scade *abs(sem_op)* din *semval*
    - conditia garanteaza ca semval ramane >= 0 !
    - la SEM_UNDO, *abs(sem_op)* se adauga la valoarea de ajustare a semaforului pt acest proces
  - *semval* < *abs(sem_op)* => resursele nu sunt disponibile
    - pt IPC_NOWAIT, *semop* se intoarce cu eroare setata pe EAGAIN
    - fara IPC_NOWAIT, *semncnt* se incrementeaza si procesul apelant se suspenda pana cand:

    i) *semval* >= *abs(sem_op)*, *semncnt* se decrementeaza si se scade *abs(sem_op)* din *semval;* la SEM_UNDO, *abs(sem_op)* se adauga la valoarea de ajustare a semaforului pt acest proces

    ii) semaforul este sters din sistem => *semop* intoarce ERMID

    iii) procesul primeste un semnal si handlerul de semnal se intoarce => *semncnt* se decrementeaza si *semop* se intoarce cu EINTR

# Semnificatia sem_op (cont.)

- valoare nula
  - daca *semval* == 0, *semop* se intoarce imediat
  - daca *semval* != 0
    - pt IPC_NOWAIT, *semop* intoarce EAGAIN
    - fara IPC_NOWAIT, *semzcnt* este incrementata si procesul apelant se suspenda pana cand

    *i)* *semval* devine 0; *semzcnt* se decrementeaza

    ii) semaforul este sters din sistem => *semop* intoarce ERMID

    iii) procesul primeste un semnal si handlerul de semnal se intoarce => *semzcnt* se decrementeaza si *semop* se intoarce cu EINTR

# Ajustarea semaforului la *exit*

- cand specificam SEM_UNDO si alocam resurse (*sem_op* < 0) kernelul memoreaza nr resurselor alocate, i.e. *abs(sem_op)*
- la terminarea procesului, kernelul verifica daca exista ajustari de facut pt semafoarele procesului si le aplica
- daca setam valoarea semaforului cu *semctl* si SETVAL/SETALL valoarea de ajustare e setata pe 0 pt toate procesele

# Memorie partajata

- permite mai multor procese sa partajeze o regiune de memorie
- cea mai rapida forma de IPC pentru ca nu e nevoie de copierea datelor
- dezavantaj: procesele care partajeaza regiunea de memorie trebuie sa se sincronizeze
  - accesele la regiunea de memorie partajata trebuie executate in sectiune critica
- creare

*int shmget(key_t key, int size, int flag);*

- *key* generat cu IPC_PRIVATE sau *ftok*
- *size* = dimensiunea regiunii de memorie partajata (*size* = 0, acces la o regiune de memorie partajata existenta)

# Atasarea memoriei partajate

*int shmat(int shmid, void *addr, int flag);*

- adresa din proces la care se ataseaza regiunea de memorie depinde de valorile ultimilor doi parametri de apel

1) *addr* = 0, kernelul ataseaza regiunea de memorie la prima adresa disponibila (metoda recomandata)
2) *addr* != 0 si SHM_RND nu apare in flag => regiunea de memorie este atasata la adresa *addr*
3) *addr* != 0 si SHM_RND setat => regiunea se ataseaza la adresa *addr* rotunjita la urmatorul multiplu al SHMLBA (*low boundary address multiple*)

- *flag* = SHM_RDONLY atasarea se face RO, altfel RW
- valoarea de retur: este adresa din proces la care a fost atasata regiunea (-1 in caz de eroare)

# Detasarea memoriei partajate

*int shmdt(void *addr);*

- Obs: detasarea nu sterge ID-ul memoriei partajate si nici structura de date aferenta
  - supravietuiesc pana cand se sterge zona de memorie partajata
- *addr* este valoarea returnata de un apel *shmat* anterior

# Operatii cu memorie partajata

- executate cu *shmctl*

*int shmctl(int shmid, int cmd, struct shmid_ds *buf);*

Valori *cmd*: IPC_STAT – citeste in *buf* structura de date asociata in kernel regiunii de memorie

IPC_SET – seteaza permisiunile (UID/GID si *mode* in structura de data asociata in kernel regiunii de memorie)

IPC_RMID – sterge regiunea de memorie din sistem imediat, a.i orice operatie subsecventa de atasare esueaza

Obs: totusi regiunea de memorie nu dispare din kernel pana cand ultimul proces atasat la ea nu a terminat (sau se detaseaza de ea)

SHM_LOCK/SHM_UNLOCK - *pinned down memory*, memoria nu poate fi swappata pe disc (doar *root*-ul poate executa operatia de *pin down*)