Programare concurentă în C (V) :

Comunicaţia inter-procese, partea I-a: Comunicaţia prin canale interne

Cristian Vidraşcu

vidrascu@info.uaic.ro

Sumar

- Introducere
- Canale interne. Primitiva `pipe()`
- Exemple: Comunicaţie între două procese
- Caracteristici şi restricţii ale canalelor interne
- Comportamentul neblocant

Introducere

Tipuri de comunicaţie între procese:

- comunicaţia prin memorie partajată
- comunicaţia prin schimb de mesaje
  - *comunicaţie locală*
    - canale interne
    - canale externe
  - *comunicaţie la distanţă*
    - *socket*-uri

Introducere (cont.)

Un *canal de comunicaţie* UNIX, sau *pipe*, este o “conductă” prin care pe la un capăt se scriu mesajele (ce constau în şiruri de octeţi), iar pe la celălalt capăt acestea sunt citite – deci este vorba despre o structură de tip coadă, adică o listă FIFO (*First-In,First-Out*).
Această “conductă” FIFO poate fi folosită pentru comunicare de către două sau mai multe procese, pentru a transmite date de la unul la altul.
Canalele de comunicaţie UNIX se împart în două subcategorii:

- *canale interne*: aceste “conducte” sunt create în memoria internă a sistemului UNIX respectiv
- *canale externe*: aceste “conducte” sunt fişiere de un tip special, numit *fifo*, deci sunt păstrate în sistemul de fişiere (aceste fişiere *fifo* se mai numesc şi *pipe*-uri cu nume)

Canale interne. Primitiva `pipe`

Un *canal intern* este un canal de comunicaţie aflat în memorie, prin care pot comunica local două (sau mai multe) procese.

Observaţie: deoarece canalele interne sunt *anonime* (*i.e.*, nu au nume), pot fi utilizate pentru comunicaţie doar de către procese “înrudite” prin apeluri `fork`/`exec`.

Canale interne. Primitiva `pipe`

Crearea unui canal intern se face cu ajutorul primitivei `pipe`.

Interfaţa acestei funcţii:

```
int pipe(int *p)
```

- *p* = parametrul efectiv de apel trebuie să fie un tablou `int[2]` ce va fi actualizat de funcţie astfel:
  - `p[0]` va fi descriptorul de fişier deschis pentru capătul de citire al canalului
  - `p[1]` va fi descriptorul de fişier deschis pentru capătul de scriere al canalului
- valoarea returnată este `0`, în caz de succes, sau `-1`, în caz de eroare.

Canale interne. Primitiva `pipe`

Crearea unui canal intern se face cu ajutorul primitivei `pipe`.
Interfaţa acestei funcţii:

```
int pipe(int *p)
```

Efect: în urma execuţiei primitivei `pipe` se creează un canal intern şi este deschis la ambele capete – în citire la capătul referit prin `p[0]`, respectiv în scriere la capătul referit prin `p[1]`.

După crearea unui canal intern, scrierea în acest canal şi citirea din el se efectuează la fel ca pentru fişierele obişnuite, cu apelurile `read` şi `write`, prin intermediul celor doi descriptori `p[0]` şi `p[1]`.

Canale interne. Primitiva `pipe`

Observaţie importantă: pentru ca două (sau mai multe) procese să poată folosi un canal intern pentru a comunica între ele, acele procese trebuie sa aibă la dispoziţie cei doi descriptori *p*`[0]` si *p*`[1]` obţinuţi prin crearea canalului. Deci procesul care a creat canalul prin apelul `pipe`, va trebui să le “transmită” cumva celuilalt proces.

Canale interne. Primitiva `pipe`

Observaţie importantă: pentru ca două (sau mai multe) procese să poată folosi un canal intern pentru a comunica între ele, acele procese trebuie sa aibă la dispoziţie cei doi descriptori *p*`[0]` si *p*`[1]` obţinuţi prin crearea canalului. Deci procesul care a creat canalul prin apelul `pipe`, va trebui să le “transmită” cumva celuilalt proces.

De exemplu, în cazul când se doreşte să se utilizeze un canal intern pentru comunicarea între două procese de tipul părinte-fiu, atunci este suficient să se apeleze primitiva `pipe` de creare a canalului **înaintea** apelului primitivei `fork` de creare a procesului fiu. În acest fel în procesul fiu avem la dispoziţie cei doi descriptori necesari pentru comunicare prin intermediul acelui canal intern.
La fel se procedează şi în cazul apelului primitivelor `exec` (deoarece descriptorii de fişiere deschise se moştenesc prin `exec`).

Canale interne. Primitiva `pipe`

Observaţie importantă: pentru ca două (sau mai multe) procese să poată folosi un canal intern pentru a comunica între ele, acele procese trebuie sa aibă la dispoziţie cei doi descriptori *p*`[0]` si *p*`[1]` obţinuţi prin crearea canalului. Deci procesul care a creat canalul prin apelul `pipe`, va trebui să le “transmită” cumva celuilalt proces.

Altă observaţie importantă: dacă un proces îşi închide vreunul din capetele unui canal intern, atunci nu mai are nicio posibilitate de a redeschide ulterior acel capăt al canalului.

Notă: vom vedea mai târziu că aceste două restricţii de folosire nu sunt valabile şi în cazul canalelor externe.

Exemple: Comunicaţie între 2 procese

Un prim exemplu: un program care exemplifică modul de utilizare a unui canal intern pentru comunicaţia între două procese, de tip producător–consumator.

A se vedea fişierul sursă `pipe-ex1.c` .

Efectul acestui program: procesul tată citeşte o secvenţă de caractere de la tastatură, secvenţă terminată cu combinaţia de taste `CTRL+D` (*i.e.*, caracterul `EOF` in `UNIX`), şi le transmite procesului fiu, prin intermediul canalului, doar pe acelea care sunt litere mici. Iar fiul citeşte din canal caracterele trasmise de procesul părinte şi le afişează pe ecran.

Observaţie: programul foloseşte primitivele `read` si `write` (*i.e.*, funcţiile I/O de nivel scăzut, ne-*buffer*-izate) pentru a citi din canal, respectiv pentru a scrie în canal.

Exemple: Comunicaţie între 2 procese

Evident, se pot folosi şi funcţiile I/O de nivel înalt pentru comunicaţia prin intermediul canalelor (în acest caz este necesară conversia descriptorilor de fişiere de la tipul `int` la tipul `FILE*`, cu ajutorul funcţiei de bibliotecă `fdopen`).

Un al doilea exemplu: un alt program care exemplifică modul de utilizare a unui canal intern pentru comunicaţia între două procese, de tip producător–consumator, cu observaţia că programul foloseşte funcţiile de bibliotecă `fscanf` şi `fprintf` (*i.e.*, funcţiile I/O de nivel înalt, *buffer*-izate) pentru a citi din canal, respectiv pentru a scrie în canal.

A se vedea fişierul sursă `pipe-ex2.c` .

Efectul acestui program: procesul tată citeşte un şir de numere de la tastatură, şir terminat cu combinaţia de taste `CTRL+D` (*i.e.*, caracterul `EOF` in `UNIX`), şi le transmite procesului fiu, prin intermediul canalului. Iar procesul fiu citeşte din canal numerele trasmise de procesul părinte şi le afişează pe ecran.

Caracteristici şi restricţii ale canalelor interne

- Canalul intern este un canal unidirecţional, adică pe la capătul `p[1]` se scrie, iar pe la capătul `p[0]` se citeşte.
  Însă toate procesele (ce au acces la acel *pipe*) pot să scrie la capătul `p[1]`, şi să citească la capătul `p[0]`.
- Unitatea de informaţie pentru canalul intern este octetul.
  Cu alte cuvinte, cantitatea minimă de informaţie ce poate fi scrisă în canal, respectiv citită din canal, este de 1 octet.
- Capacitatea canalului intern este limitată la o anumită dimensiune maximă (4 Ko, 16 Ko, etc.), ce poate diferi de la o versiune de `UNIX` la alta.

Caracteristici şi restricţii ale canalelor interne

- Canalul intern funcţionează ca o coadă, adică o listă FIFO (*First-In, First-Out*), deci citirea din canal se face cu distrugerea (*i.e.*, consumul) din canal a informaţiei citite.

  Aşadar, citirea dintr-un canal diferă de citirea din fişiere obişnuite, pentru care citirea se face fără consumul informaţiei din fişier.

  În plus, pentru canale nu există noţiunea de *offset* (*i.e.*, poziţie curentă), ca în cazul fişierelor obişnuite, la care se efectuează operaţia de citire/scriere.

Caracteristici şi restricţii ale canalelor interne (cont.)

- Citirea dintr-un canal intern (cu primitiva `read`) funcţionează în felul următor:
  - Apelul `read` va citi din canal şi va returna imediat, fără să se blocheze, numai dacă mai este suficientă informaţie în canal, iar în acest caz valoarea returnată reprezintă numărul de octeţi citiţi din canal.
  - Altfel, dacă canalul este gol, sau nu conţine suficientă informaţie, apelul de citire `read` va rămâne blocat până când va avea suficientă informaţie în canal pentru a putea citi cantitatea de informaţie specificată, ceea ce se va întâmpla în momentul când un alt proces va scrie în canal.

Caracteristici şi restricţii ale canalelor interne (cont.)

- Citirea dintr-un canal intern (cu primitiva `read`) funcţionează în felul următor:
  - Alt caz de excepţie la citire, pe lângă cazul golirii canalului: dacă un proces încearcă să citească din canal şi nici un proces nu mai este capabil să scrie în canal vreodată (deoarece toate procesele şi-au închis deja capătul de scriere), atunci apelul `read` returnează imediat valoarea `0` corespunzătoare faptului că a citit `EOF` din canal.

    În concluzie, pentru a se putea citi `EOF` din canal, trebuie ca mai întâi toate procesele să închidă canalul în scriere (adică să închidă descriptorul *p*`[1]`).

Observaţie: la fel se comportă la citirea din canale interne şi funcţiile de citire de nivel înalt (`fread`, `fscanf`, etc.), doar că acestea lucrează *buffer*-izat.

Caracteristici şi restricţii ale canalelor interne (cont.)

- Scrierea într-un canal intern (cu primitiva `write`) funcţionează în felul următor:
  - Apelul `write` va scrie în canal şi va returna imediat, fără să se blocheze, numai dacă mai este suficient spaţiu liber în canal, iar în acest caz valoarea returnată reprezintă numărul de octeti efectiv scrişi în canal (care poate să nu coincidă întotdeauna cu numărul de octeti ce se doreau a se scrie, căci pot apare erori I/O).
  - Altfel, dacă canalul este plin, sau nu conţine suficient spaţiu liber, apelul de scriere `write` va rămâne blocat până când va avea suficient spaţiu liber în canal pentru a putea scrie informaţia specificată ca argument, ceea ce se va întâmpla în momentul când un alt proces va citi din canal.

Caracteristici şi restricţii ale canalelor interne (cont.)

- Scrierea într-un canal intern (cu primitiva `write`) funcţionează în felul următor:
  - Alt caz de excepţie la scriere, în afara umplerii canalului: dacă un proces încearcă să scrie în canal şi nici un proces nu mai este capabil să citească din canal vreodată (deoarece toate procesele şi-au închis deja capătul de citire), atunci sistemul va trimite acelui proces semnalul `SIGPIPE`, ce cauzează terminarea forţată a procesului, fără a afişa însă vreun mesaj de eroare. (Observaţie: versiunile mai vechi de kernel Linux afişau mesajul de eroare "Broken pipe".)

Observaţie: la fel se comportă la scrierea în canale interne şi funcţiile de scriere de nivel înalt (`fwrite`, `fprintf`, etc.), doar că acestea lucrează *buffer*-izat.

Notă: modul de lucru *buffer*-izat al funcţiilor de scriere din biblioteca standard de I/O din C, poate cauza uneori erori dificil de depistat, datorate neatenţiei programatorului, care poate uita să forţeze "golirea" *buffer*-ului în canal cu ajutorul funcţiei `fflush`, imediat după apelul funcţiei de scriere propriu-zise.

Comportamentul neblocant

Cele afirmate mai devreme, despre blocarea apelurilor de citire sau de scriere în cazul canalului gol, respectiv plin, corespund comportamentului implicit, de tip **blocant**, al canalelor interne.

Acest comportament implicit poate fi modificat într-un comportament de tip **neblocant**, situaţie în care apelurile de citire sau de scriere nu mai rămân blocate în cazul canalului gol, respectiv plin, ci returnează imediat valoarea `-1`, şi setează corespunzător variabila `errno`.

Comportamentul neblocant (cont.)

Modificarea comportamentului implicit în comportament **neblocant** se realizează prin setarea atributului `O_NONBLOCK` pentru descriptorul corespunzător acelui capăt al canalului intern pentru care se doreşte modificarea comportamentului, cu ajutorul primitivei `fcntl`.

Spre exemplu, apelul

```
fcntl(p[1],F_SETFL,O_NONBLOCK);
```

va seta atributul `O_NONBLOCK` pentru capătul de scriere al canalului intern referit de variabila *p*.

Temă: scrieţi un program prin care să determinaţi capacitatea canalelor interne pe sistemul `Linux` pe care lucraţi.

Bibliografie obligatorie

Cap.5, §5.1 şi §5.2 din manualul, în format PDF, accesibil din pagina disciplinei “Sisteme de operare”:

- `http://profs.info.uaic.ro/~vidrascu/SO/books/ManualID-SO.pdf`

Programele demonstrative amintite pe parcursul acestei prezentări pot fi descărcate de la adresa următoare:

- `http://profs.info.uaic.ro/~vidrascu/SO/cursuri/C-programs/pipe/`