Programare concurentă în C (VI) :

Comunicaţia inter-procese, partea a II-a: Comunicaţia prin canale externe

Cristian Vidraşcu

vidrascu@info.uaic.ro

Sumar

- Introducere
- Canale externe (i.e. fişiere *fifo*)
- Caracteristici şi restricţii ale canalelor externe
- Comportamentul neblocant
- Deosebiri între canalele externe şi cele interne
- Aplicaţie: implementarea unui semafor
- Aplicaţie: programe de tip client/server
- Şabloane de comunicaţie între procese

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 externe

Un *canal extern* este un canal de comunicaţie prin care pot comunica local două (sau mai multe) procese, comunicaţia realizându-se în acest caz printr-un fişier de tip *fifo*.

Notă: spaţiul de stocare al informaţiilor conţinute acest tip de fişiere este tot în memoria principală, nu pe disc (practic conţinutul unui fişier fifo este tot o coadă FIFO aflată în memorie şi gestionată de SO, la fel ca şi în cazul canalelor anonime).

Observaţie: deoarece canalele externe nu sunt *anonime* (*i.e.*, au nume prin care pot fi referite), pot fi utilizate pentru comunicaţie de către orice procese care cunosc numele fişierului *fifo* respectiv, deci nu mai avem restricţia de la canale interne, aceea că procesele trebuiau să fie “înrudite” prin `fork`/`exec`.

Canale externe. Primitiva `mkfifo`

Crearea unui canal extern se face cu ajutorul primitivei `mkfifo`.
Interfaţa acestei funcţii:

`int mkfifo(char *`*nume*`, int` *permisiuni*`)`

- *nume* = numele fişierului (de tip *fifo*) ce va fi creat
- *permisiuni* = permisiunile pentru fişierul ce va fi creat
- valoarea returnată este `0`, în caz de succes, sau `-1`, în caz de eroare.

Efect: în urma execuţiei primitivei `fifo` se creează un canal extern (dar fără a fi deschis la ambele capete).

Canale externe. Primitiva `mkfifo`

Crearea unui canal extern se face cu ajutorul primitivei `mkfifo`.

Exemplu de creare a unui fişier *fifo*: a se vedea fişierul sursă `mkfifo-ex.c` .

Notă: crearea unui fişier *fifo* se mai poate face cu ajutorul primitivei `mknod` apelată cu *flag*-ul `S_IFIFO`. De asemenea, poate fi creat şi direct de la prompterul *shell*-ului, cu comenzile `mkfifo` sau `mknod`.

Canale externe (cont.)

După crearea unui canal extern, modul de utilizare al acestuia este similar ca la fişierele obişnuite:
mai întâi se deschide fişierul, apoi se scrie în el şi/sau se citeşte din el, iar la sfârşit se închide fişierul.

Aşadar, operaţiile asupra canalelor *fifo* se vor face fie cu primitivele I/O de nivel scăzut (*i.e.*, `open`, `read`, `write`, `close`), fie cu funcţiile I/O de nivel înalt din biblioteca standard de I/O din C (*i.e.*, `fopen`, `fread`/`fscanf`, `fwrite`/`fprintf`, `fclose`, ş.a.).

Canale externe (cont.)

La fel ca pentru fişiere obşnuite, deschiderea unui fişier *fifo* se poate face într-unul din următoarele trei moduri posibile:

- *read & write* (deschiderea ambelor capete ale canalului)
- *read-only* (deschiderea doar a capătului de citire)
- *write-only* (deschiderea doar a capătului de scriere)

modul fiind specificat prin parametrul transmis funcţiei de deschidere.

Observaţie importantă: implicit, deschiderea se face în mod *blocant*, *i.e.* o deschidere *read-only* trebuie să se "sincronizeze" cu una *write-only*. Cu alte cuvinte, dacă un proces încearcă să deschidă un capăt al canalului extern, apelul funcţiei de deschidere rămâne blocat (*i.e.*, funcţia nu returnează) până când un alt proces va deschide celălalt capăt al canalului.

Canale externe (cont.)

Perioada de retenţie a informaţiei stocate într-un canal:

Spre deosebire de fişierele obşnuite, care păstrează informaţia scrisă în ele pe perioadă nedeterminată (mai precis, până la o eventuală operaţie de modificare/ştergere), în cazul unui fişier *fifo* informaţia scrisă în canal se păstrează doar din momentul scrierii şi până în momentul când atât procesul care a scris acea informaţie, cât şi orice alt proces ce accesa acel canal, termină accesul la acel canal (închizându-şi capetele canalului), iar aceasta numai dacă informaţia nu este consumată mai devreme, prin citire.

A se vedea fişierul sursă `testare_retentie_fifo.c` .

Caracteristici şi restricţii ale canalelor externe

- Canalul extern este un canal unidirecţional, adică pe la un capăt se scrie, iar pe la capătul opus se citeşte.
  Însă toate procesele (ce au acces la acel *fifo*) pot să scrie la capătul de scriere, şi să citească la capătul de citire.
- Unitatea de informaţie pentru canalul extern 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 extern 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 externe

- Canalul extern 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 fişier *fifo* diferă de citirea din fişiere obişnuite, pentru care citirea se face fără consumul informaţiei din fişier.

  În plus, pentru fişiere *fifo* 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 externe (cont.)

- Citirea dintr-un canal extern (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 externe (cont.)

- Citirea dintr-un canal extern (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 (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 corespunzător capătului de scriere).

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

Caracteristici şi restricţii ale canalelor externe (cont.)

- Scrierea într-un canal extern (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 externe (cont.)

- Scrierea într-un canal extern (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 externe 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 externe.

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`.
Iar o deschidere neblocantă a unuia din capetele canalului va reuşi imediat, fără să mai aştepte ca alt proces să deschidă celălat capăt.

Comportamentul neblocant (cont.)

Modificarea comportamentului implicit în comportament **neblocant** se realizează prin setarea atributului `O_NONBLOCK` pentru descriptorul corespunzător acelui capăt al canalului extern pentru care se doreşte modificarea comportamentului.

Aceasta se poate face fie direct la deschiderea canalului, fie după deschidere cu ajutorul primitivei `fcntl`. Spre exemplu, apelul

fd_out = `open(“`*canal_fifo*`”, O_WRONLY|O_NONBLOCK);`

va seta la deschidere atributul `O_NONBLOCK` pentru capătul de scriere al canalului specificat. Sau se poate seta după deschidere, cu apelul

`fcntl(`*fd_out*`,F_SETFL,O_NONBLOCK);`

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

Deosebiri ale canalelor externe faţă de cele interne

- Funcţia de creare a unui canal extern nu produce şi deschiderea automată a celor două capete, acestea trebuie după creare să fie deschise explicit prin apelul unei funcţii de deschidere a unui fişier.
- Un canal extern poate fi deschis, la oricare din capete, de orice proces, indiferent dacă acel proces are sau nu vreo legătură de rudenie (prin `fork`/`exec`) cu procesul care a creat canalul extern. Aceasta este posibil deoarece un proces trebuie doar să cunoască numele fişierului *fifo* pe care doreşte să-l deschidă, pentru a-l putea deschide. Bineînţeles, procesul respectiv mai trebuie să aibă şi drepturi de acces pentru acel fişier *fifo*.
- După ce un proces închide un capăt al unui canal *fifo*, acel proces poate redeschide din nou acel capăt pentru a face alte operaţii I/O asupra sa.

Aplicaţie: implementarea unui semafor

Cum am putea implementa un semafor folosind canale *fifo* ?

Ideea: iniţializarea semaforului ar consta în crearea unui fişier *fifo* de către un proces cu rol de *supervizor* (poate fi oricare dintre procesele ce vor folosi acel semafor, sau poate fi un proces separat). Iniţial acest proces *supervizor* va scrie în canal 1 octet oarecare, dacă e vorba de un semafor binar (sau n octeţi oarecare, dacă e vorba de un semafor general n-ar), şi va păstra deschise ambele capete ale canalului pe toată durata de execuţie a proceselor ce vor folosi acel semafor (cu scopul de a nu se pierde pe parcurs informaţia din canal datorită inexistenţei la un moment dat pentru fiecare capăt a măcar unui proces care să-l aibă deschis).

Aplicaţie: implementarea unui semafor

Cum am putea implementa un semafor folosind canale *fifo* ?

Operaţia `wait` va consta în citirea unui octet din fişierul *fifo*. Mai precis, întâi se va face deschiderea lui, urmată de citirea efectivă a unui octet, şi apoi eventual închiderea fişierului.
Operaţia `signal` va consta în scrierea unui octet în fişierul *fifo*. Mai precis, întâi se va face deschiderea lui, urmată de scrierea efectivă a unui octet, şi apoi eventual închiderea fişierului.

Observaţii: i) citirea se va face în modul implicit, *blocant*, ceea ce va asigura aşteptarea procesului la punctul de intrare în zona sa critică în situaţia cînd semaforul este “pe roşu”, adică dacă canalul *fifo* este gol.
ii) scrierea nu se va putea bloca (cu condiţia ca n-ul semaforului general să nu depăşească capacitatea unui canal extern).

Aplicaţie: programe de tip client/server

O *aplicaţie de tip client/server* este compusă din două componente:

- **serverul**: este un program care dispune de un anumit număr de *servicii* (*i.e.* funcţii/operaţii), pe care le pune la dispoziţia clienţilor.
- **clientul**: este un program care "interoghează" serverul, solicitându-i *efectuarea unui serviciu* (dintre cele puse la dispoziţie de acel server).

Browser-ele pe care le folosiţi pentru a naviga pe INTERNET sunt un exemplu de program client, care se conectează la un program server, numit *server de web*, solicitându-i transmiterea unei pagini *web*, care apoi este afişată în fereastra grafică a *browser*-ului.

Aplicaţie: programe de tip client/server

Folosirea unei aplicaţii de tip client-server se face în felul următor:

Programul server va fi rulat în *background*, şi va sta în aşteptarea cererilor din partea clienţilor, putând servi mai mulţi clienţi simultan.
Iar clienţii vor putea fi rulaţi mai mulţi simultan (din acelaşi cont sau din conturi utilizator diferite), şi se vor conecta la serverul rulat în *background*.
Deci vom putea avea la un moment dat mai multe procese client, care încearcă, fiecare independent de celelalte, să folosească serviciile puse la dispoziţie de procesul server.

Observaţie: în realitate, programul server este rulat pe un anumit calculator, iar clienţii pe diverse alte calculatoare, conectate la INTERNET, comunicaţia realizându-se folosind *socket*-uri, prin intermediul reţelelor de calculatoare. Însă putem simula aceasta folosind comunicaţie prin canale externe şi executând toate procesele (*i.e.*, serverul şi clienţii) pe un acelaşi calculator, eventual din conturi utilizator diferite.

Aplicaţie: programe de tip client/server

Tipurile de servere existente în realitate, d.p.d.v. al servirii “simultane” a mai multor clienţi, se împart în două categorii:

- server *iterativ*

  Cât timp durează efectuarea unui serviciu (*i.e.*, rezolvarea unui client), serverul este blocat: nu poate răspunde cererilor venite din partea altor clienţi. Deci nu poate rezolva mai mulţi clienţi în acelaşi timp!

- server *concurent*

  Pe toată durata de timp necesară pentru efectuarea unui serviciu (*i.e.*, rezolvarea unui client), serverul nu este blocat, ci poate răspunde cererilor venite din partea altor clienţi. Deci poate rezolva mai mulţi clienţi în acelaşi timp!

Aplicaţie: programe de tip client/server

Detalii legate de implementare:

- Pentru comunicarea între procesele client şi procesul server este necesar să se utilizeze, drept canale de comunicaţie, fişiere *fifo*.
  Atenţie: nu se pot folosi *pipe*-uri interne, deoarece procesul server şi procesele clienţi nu sunt înrudite (prin `fork`/`exec`).
- Mai mult, trebuie avut grijă la gestiunea drepturilor de acces la fişierele *fifo* folosite pentru comunicaţie, astfel încât să se poată rula procesele client simultan, din diferite conturi utilizator.

Aplicaţie: programe de tip client/server

Detalii legate de implementare:

- Un alt aspect legat tot de comunicaţie: serverul nu cunoaşte în avans clienţii ce se vor conecta la el pentru a le oferi servicii, în schimb clientul trebuie să cunoască serverul la care se va conecta pentru a beneficia de serviciul oferit de el.

  Ce înseamnă aceasta d.p.d.v. practic?

  Serverul va crea un canal *fifo* cu un nume fixat, cunoscut în programul client, şi va aştepta sosirea informaţiilor pe acest canal. Un client oarecare se va conecta la acest canal *fifo* cunoscut şi va transmite informaţii de identificare a sa, care vor fi folosite ulterior pentru realizarea efectivă a comunicaţiilor implicate de serviciul solicitat (s-ar putea să fie nevoie de canale suplimentare, particulare pentru acel client, ca să nu se amestece comunicaţiile destinate unui client cu cele destinate altui client conectat la server în acelaşi timp cu primul).

Aplicaţie: programe de tip client/server

Detalii legate de implementare:

- Pentru implementarea unui server de tip iterativ este suficient un singur proces `UNIX`. În schimb, pentru implementarea unui server de tip concurent este nevoie de mai multe procese `UNIX`: un proces *master*, care aşteaptă sosirea cererilor din partea clienţilor, şi la fiecare cerere sosită, el va crea un nou proces fiu, un *slave* care va fi responsabil cu rezolvarea propriu-zisă a clientului respectiv, iar *master*-ul va relua imediat aşteptarea unei noi cereri, fără să aştepte terminarea procesului fiu.

Temă: implementaţi un joc *multi-player* “în reţea”.

Şabloane de comunicaţie între procese

După numărul de procese “scriitori” şi “cititori” ce utilizează un canal (intern sau extern) pentru a comunica între ele, putem diferenţia următoarele şabloane de comunicaţie inter-procese:

- comunicaţie *unul la unul*: canalul este folosit de un singur proces “scriitor” pentru a transmite date unui singur proces “cititor”
- comunicaţie *unul la mulţi*: canalul este folosit de un singur proces “scriitor” pentru a transmite date mai multor procese “cititori”
- comunicaţie *mulţi la unul*: canalul e folosit de mai multe procese “scriitori” pentru a transmite date unui singur proces “cititor”
- comunicaţie *mulţi la mulţi*: canalul e folosit de mai multe procese “scriitori” pentru a transmite date mai multor procese “cititori”

Şabloane de comunicaţie (cont.)

Comunicaţia *unul la unul* este cel mai simplu caz, neridicând probleme deosebite de implementare.
Notă: exemplele de programe date anterior în lecţia despre canale interne, se încadrează în acest şablon de comunicaţie.

Celelalte cazuri ridică anumite probleme, datorate accesului posibil concurent al mai multor procese la unul (sau ambele) dintre capetele canalului, probleme de care trebuie să se ţină cont la implementarea lor.

Să trecem în revistă pe rând aceste probleme . . .

Şablonul unul la mulţi

Factori ce pot genera anumite probleme (e.g. “coruperea” mesajelor):

- *lungimea* mesajelor:
  - mesaje de lungime *constantă*

    Nu ridică probleme deosebite – fiecare mesaj poate fi citit *atomic* (*i.e.*, dintr-o dată, printr-un singur apel `read`).
  - mesaje de lungime *variabilă*

Şablonul unul la mulţi

Factori ce pot genera anumite probleme (e.g. “coruperea” mesajelor):

- *lungimea* mesajelor:
  - mesaje de lungime *constantă*
  - mesaje de lungime *variabilă*

    Pot apare probleme deoarece mesajele nu mai pot fi citite *atomic*.
    Soluţia este folosirea mesajelor formatate astfel:

    MESAJ = HEADER + MESAJUL PROPRIU-ZIS ,

    header-ul fiind un mesaj de lungime fixă ce conţine lungimea mesajului propriu-zis.
    Protocol utilizat: sunt necesare 2 apeluri `read` pentru a citi un mesaj în întregime, de aceea trebuie garantat accesul exclusiv la canal (folosind, de exemplu, blocaje pe fişiere).

Şablonul unul la mulţi

Factori ce pot genera anumite probleme (e.g. “coruperea” mesajelor):

- *destinatarul* mesajelor:
  - mesaje cu destinatar *oarecare*

    Nu ridică probleme deosebite – fiecare mesaj poate fi citit şi prelucrat de oricare dintre procesele “cititori”.
  - mesaje cu destinatar *specificat*

Şablonul unul la mulţi

Factori ce pot genera anumite probleme (e.g. “coruperea” mesajelor):

- *destinatarul* mesajelor:
  - mesaje cu destinatar *oarecare*
  - mesaje cu destinatar *specificat*

    Trebuie asigurat faptul că mesajul este citit exact de către “cititorul” căruia îi era destinat. Soluţia – mesaje formatate:

    MESAJ = HEADER + MESAJUL PROPRIU-ZIS ,

    header-ul conţinând un identificator al destinatarului.

    Notă: pentru citire se aplică tehnicile discutate la mesaje de lungime variabilă.

    Protocol utilizat: dacă un “cititor” a citit un mesaj care nu-i era destinat lui, îl va scrie înapoi în canal, şi apoi va face o pauză aleatoare înainte de a încerca să citească din nou din canal.

Şablonul mulţi la unul

Factori ce pot genera anumite probleme (e.g. “coruperea” mesajelor):

- *lungimea* mesajelor:
  - mesaje de lungime *constantă*

    Nu ridică probleme deosebite – fiecare mesaj poate fi scris *atomic* (*i.e.*, dintr-o dată, printr-un singur apel `write`).
  - mesaje de lungime *variabilă*

Şablonul mulţi la unul

Factori ce pot genera anumite probleme (e.g. “coruperea” mesajelor):

- *lungimea* mesajelor:
  - mesaje de lungime *constantă*
  - mesaje de lungime *variabilă*

    Trebuie indicată “cititorului” lungimea fiecărui mesaj. Soluţia este folosirea mesajelor formatate astfel:

    MESAJ = HEADER + MESAJUL PROPRIU-ZIS ,

    header-ul fiind un mesaj de lungime fixă ce conţine lungimea mesajului propriu-zis.

    Nu ridică probleme deosebite – fiecare mesaj poate fi scris printr-un singur apel `write`, deci nu mai trebuie garantat accesul exclusiv la canal.

Şablonul mulţi la unul

Factori ce pot genera anumite probleme (e.g. “coruperea” mesajelor):

- *expeditorul* mesajelor:
  - mesaje cu expeditor *oarecare*

    Nu ridică probleme deosebite – fiecare mesaj poate fi citit de procesul “cititor” şi prelucrat în acelaşi fel indiferent de la care dintre procesele “scriitori” provine.
  - mesaje cu expeditor *specificat*

Şablonul mulţi la unul

Factori ce pot genera anumite probleme (e.g. “coruperea” mesajelor):

- *expeditorul* mesajelor:
  - mesaje cu expeditor *oarecare*
  - mesaje cu expeditor *specificat*

    Trebuie asigurat că mesajul îi indică “cititorului” care este “scriitorul” care i l-a trimis. Soluţia – mesaje formatate:

    MESAJ = HEADER + MESAJUL PROPRIU-ZIS ,

    header-ul conţinând un identificator al expeditorului.

    Notă: scrierea mesajului astfel formatat se va face printr-un singur apel `write`, la fel ca la mesaje de lungime variabilă.

Şablonul mulţi la mulţi

Pot interveni toate categoriile de factori de la comunicaţiile *unul la mulţi* şi *mulţi la unul*:

- *lungimea* mesajelor
- *expeditorul* mesajelor
- *destinatarul* mesajelor

Tratarea acestora se face prin combinarea soluţiilor prezentate la comunicaţiile *unul la mulţi* şi *mulţi la unul*.

Observaţie: pentru simplitate, se poate prefera uneori înlocuirea unui singur canal folosit pentru comunicaţie *mulţi la unul*, cu mai multe canale folosite pentru comunicaţie *unul la unul*, câte un canal pentru fiecare proces “scriitor” existent. Similar se poate proceda şi pentru cazul *unul la mulţi*, sau pentru cazul *mulţi la mulţi*.

Bibliografie obligatorie

Cap.5, §5.3 şi §5.5 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/fifo/`