

# PROGRAMARE DE SISTEM ÎN C PENTRU PLATFORMA LINUX (V)

## Comunicația inter-procese: Canale de comunicație anonime și cu nume

Cristian Vidrașcu  
vidrascu@info.uaic.ro

Mai, 2021



# Sumar

- Introducere
- Canale anonime
- Canale cu nume (*fifo*)
- Caracteristici comune pentru ambele tipuri de canale
- Șabloane de comunicație între procese
- Aplicații ale canalelor de comunicație
- Referințe bibliografice

- Introducere
- Canale anonime**
  - Crearea lor, cu primitiva `pipe`
  - Modul de utilizare a unui canal anonim
  - Demo*: exemple de comunicație între două procese
- Canale cu nume (*fifo*)**
  - Crearea lor, cu primitiva `mkfifo`
  - Modul de utilizare a unui canal cu nume
  - Despre persistența informației dintr-un fișier *fifo*
  - Deosebiri ale canalelor cu nume față de cele anonime
- Caracteristici comune pentru ambele tipuri de canale**
  - Caracteristici și restricții ale canalelor de comunicație
  - Comportamentul implicit, de tip blocant
  - Comportamentul de tip neblocant
- Șabloane de comunicație între procese**
  - Clasificarea șabloanelor de comunicație inter-procese
  - Șablonul de comunicație *unul-la-unul*
  - Șablonul de comunicație *unul-la-multi*
  - Șablonul de comunicație *multi-la-unul*
  - Șablonul de comunicație *multi-la-multi*
- Aplicații ale canalelor de comunicație**
  - Aplicația #1: implementarea unui semafor
  - Aplicația #2: implementarea unei aplicații client/server
- Referințe bibliografice**



# Introducere

## Introducere

### [Canale anonime](#)

### [Canale cu nume \(\*fifo\*\)](#)

### [Caracteristici comune pentru ambele tipuri de canale](#)

### [Șabloane de comunicație între procese](#)

### [Aplicații ale canalelor de comunicație](#)

### [Referințe bibliografice](#)

## Tipuri de comunicație între procese:

- comunicația prin **memorie partajată** (*“shared-memory communication”*)  
e.g. prin fișiere mapate în memorie, sau mapări anonime și cu nume, ș.a.
- comunicația prin **schimb de mesaje** (*“message-passing communication”*)
  - *comunicație locală*
    - ▲ canale anonime (numite, uneori, și canale interne)
    - ▲ canale cu nume, *i.e.* fișiere *fifo* (numite, uneori, și canale externe)
  - *comunicație la distanță*
    - ▲ *socket-uri*



## Introducere (cont.)

### Introducere

#### Canale anonime

#### Canale cu nume (*fifo*)

#### Caracteristici comune pentru ambele tipuri de canale

#### Șabloane de comunicație între procese

#### Aplicații ale canalelor de comunicație

#### Referințe bibliografice

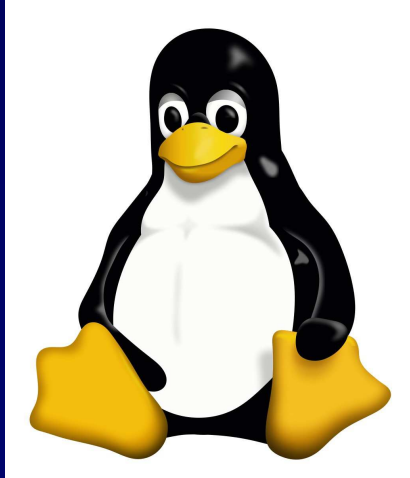
Un *canal de comunicație* UNIX, sau *pipe*, este o “conductă” prin care pe la un capăt se scriu mesajele (ce constau în secvențe de octeți), iar pe la celălalt capăt acestea sunt citite (cu extracția lor din canal) – deci practic se comportă ca o structură de tip coadă, adică o listă FIFO (*First-In, First-Out*).

*Notă:* de fapt, un *pipe* chiar este implementat de nucleul UNIX/Linux ca o listă FIFO, cu o capacitate constantă, gestionată în *kernel-space*.

**Rolul unui canal:** o asemenea “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 comunicație UNIX se împart în două subcategorii:

- ***canale anonime*:** aceste “conduce” sunt create în memoria internă a sistemului UNIX respectiv, fără niciun nume asociat lor în sistemul de fișiere;
- ***canale cu nume*:** aceste “conduce” sunt create tot în memoria internă a sistemului, dar au asociate câte un nume, reprezentat printr-un fișier de tipul special *fifo*, care este păstrat în sistemul de fișiere (din acest motiv, aceste fișiere *fifo* se mai numesc și *pipe-uri* cu nume).



# Agenda

Introducere

Canale anonime

Crearea lor, cu primitiva `pipe`  
Modul de utilizare a unui canal anonim  
*Demo*: exemple de comunicație între două procese

Canale cu nume (*fifo*)

Caracteristici comune pentru ambele tipuri de canale

Șabloane de comunicație între procese

Aplicații ale canalelor de comunicație

Referințe bibliografice

Introducere

**Canale anonime**

Crearea lor, cu primitiva `pipe`  
Modul de utilizare a unui canal anonim  
*Demo*: exemple de comunicație între două procese

**Canale cu nume (*fifo*)**

Crearea lor, cu primitiva `mkfifo`  
Modul de utilizare a unui canal cu nume  
Despre persistența informației dintr-un fișier *fifo*  
Deosebiri ale canalelor cu nume față de cele anonime

**Caracteristici comune pentru ambele tipuri de canale**

Caracteristici și restricții ale canalelor de comunicație  
Comportamentul implicit, de tip blocant  
Comportamentul de tip neblocant

**Șabloane de comunicație între procese**

Clasificarea șabloanelor de comunicație inter-procese  
Șablonul de comunicație *unul-la-unul*  
Șablonul de comunicație *unul-la-multi*  
Șablonul de comunicație *multi-la-unul*  
Șablonul de comunicație *multi-la-multi*

**Aplicații ale canalelor de comunicație**

Aplicația #1: implementarea unui semafor  
Aplicația #2: implementarea unei aplicații client/server

**Referințe bibliografice**





# Crearea lor, cu primitiva `pipe`

Introducere

Canale anonime

Crearea lor, cu primitiva `pipe`

Modul de utilizare a unui canal anonim

Demo: exemple de comunicație între două procese

Canale cu nume (*fifo*)

Caracteristici comune pentru ambele tipuri de canale

Șabloane de comunicație între procese

Aplicații ale canalelor de comunicație

Referințe bibliografice

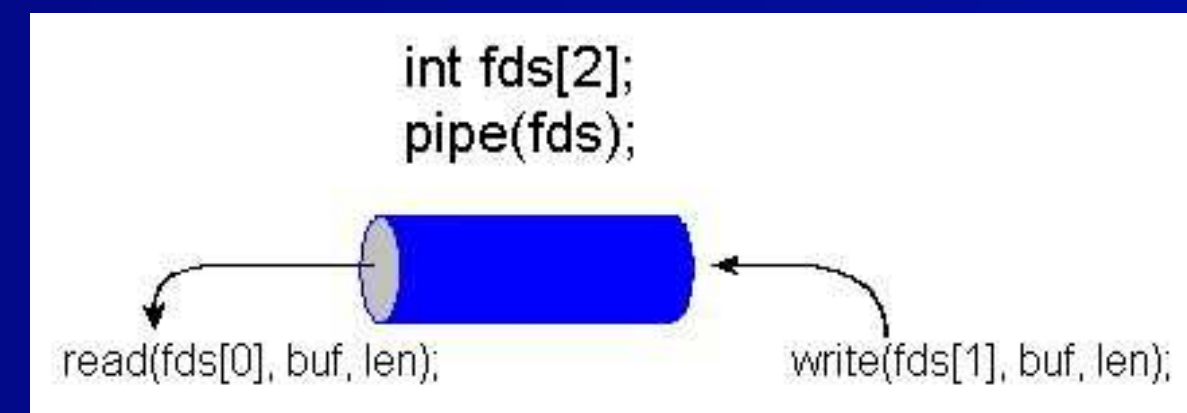
Un *canal anonim* se creează cu ajutorul primitivei `pipe`.

Interfața acestei funcții este următoarea ([5]):

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

- $p$  = parametrul efectiv de apel trebuie să fie un vector `int [2]`, care va fi actualizat de funcție în felul următor:
  - $p[0]$  va fi descriptorul de fișier deschis pentru *capătul de citire* al canalului
  - $p[1]$  va fi descriptorul de fișier deschis pentru *capătul de scriere* al canalului
- valoarea returnată este 0, în caz de succes, sau -1, în caz de eroare.

**Efect:** în urma execuției primitivei `pipe` se creează un canal anonim și *este deschis automat la ambele capete* – în citire la capătul referit prin descriptorul  $p[0]$  și, respectiv, în scriere la capătul referit prin descriptorul  $p[1]$ .





## Modul de utilizare a unui canal anonim

Introducere

Canale anonime

Crearea lor, cu primitiva `pipe`

Modul de utilizare a unui canal anonim

Demo: exemple de comunicație între două procese

Canale cu nume (*fifo*)

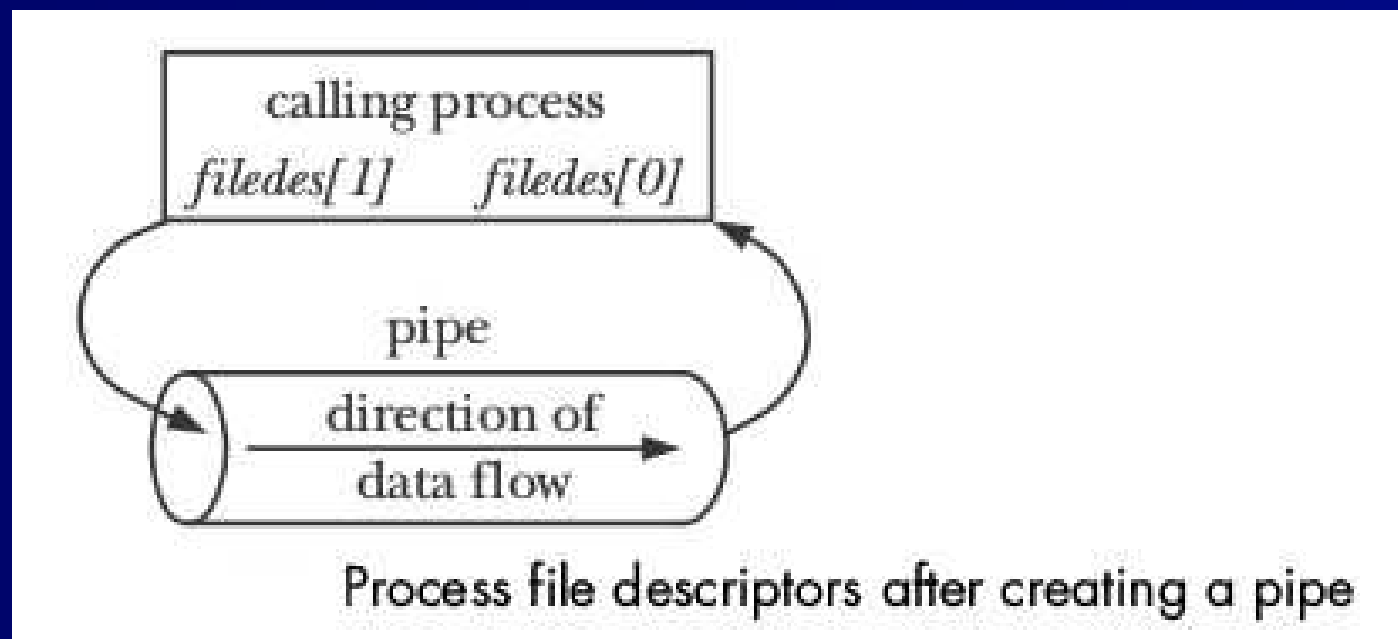
Caracteristici comune pentru ambele tipuri de canale

Șabloane de comunicație între procese

Aplicații ale canalelor de comunicație

Referințe bibliografice

După crearea unui canal anonim, folosirea sa pentru comunicația locală între două (sau mai multe) procese se face prin scrierea informației în acest canal și, respectiv, prin citirea informației din canal.



Iar scrierea în canal și respectiv citirea din canal, prin intermediul celor doi descriptori  $p[0]$  și  $p[1]$ , se efectuează la fel ca pentru fișierele obișnuite, *i.e.* folosind apelurile `read` și `write`, sau cu funcțiile I/O din biblioteca `stdio`.

\* \* \*

*Restricție importantă:*

Deoarece acest tip de canale sunt *anonime* (*i.e.*, nu au nume), pot fi utilizate pentru comunicație doar de către procese “înrudite” prin apeluri `fork/exec`.

De ce? Motivația este următoarea: ... (vezi slide-ul următor)



## Modul de utilizare a unui canal anonim (cont.)

Introducere

Canale anonime

Crearea lor, cu primitiva `pipe`

Modul de utilizare a unui canal anonim

Demo: exemple de comunicație între două procese

Canale cu nume (*fifo*)

Caracteristici comune pentru ambele tipuri de canale

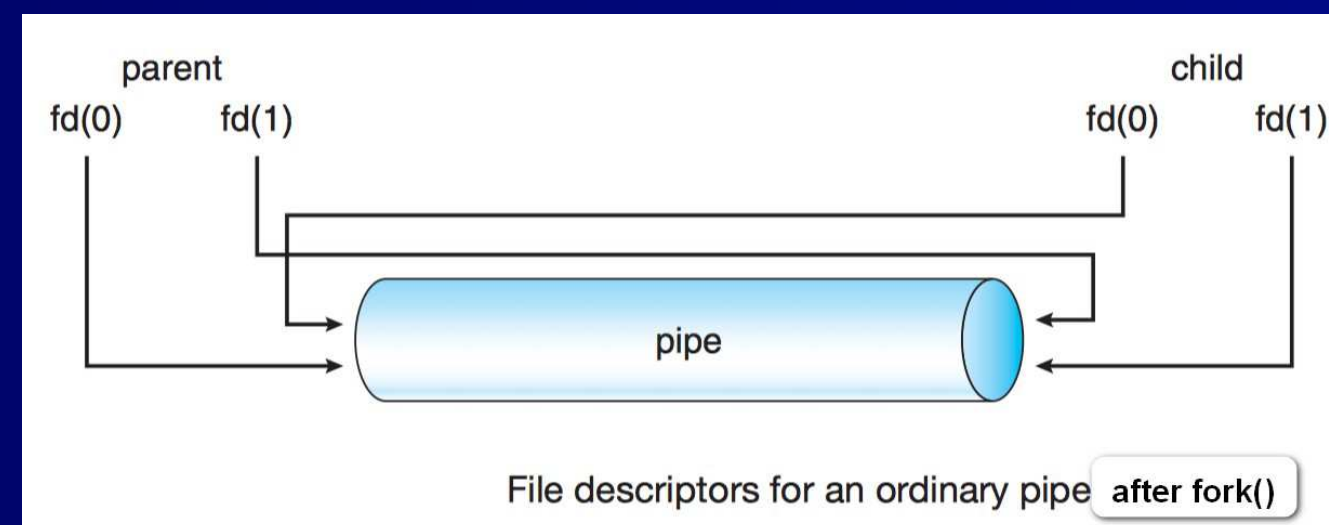
Șabloane de comunicație între procese

Aplicații ale canalelor de comunicație

Referințe bibliografice

*Motivație:* pentru ca două (sau mai multe) procese să poată folosi un canal anonim pentru a comunica între ele, acele procese trebuie să aibă la dispoziție cei doi descriptori  $p[0]$  și  $p[1]$  obținuți prin crearea canalului. Deci procesul care a creat canalul prin apelul `pipe`, va trebui să le “transmită” cumva celui alt proces.

De exemplu, în cazul când se dorește să se utilizeze un canal anonim pentru comunicarea între două procese de tipul părinte-fiu, atunci este suficient **să se apeleze primitiva `pipe` de creare a canalului înainte apelului primitivei `fork` de creare a procesului fiu**. În acest fel, prin clonare, avem la dispoziție și în procesul fiu cei doi descriptori necesari pentru comunicare prin intermediul acelui canal anonim.



*Notă:*

“Transmiterea” descriptorilor canalului are loc și în cazul apelului primitivelor `exec` (deoarece descriptorii de fișiere deschise se moștenesc prin `exec`).





# Modul de utilizare a unui canal anonim (cont.)

*Altă restricție:*  
Dacă un proces își închide vreunul dintre capetele unui canal anonim, atunci nu mai are nicio posibilitate de a redeschide ulterior acel capăt al canalului.

\* \* \*

Figura 1 – un canal anonim, imediat după apelul pipe:

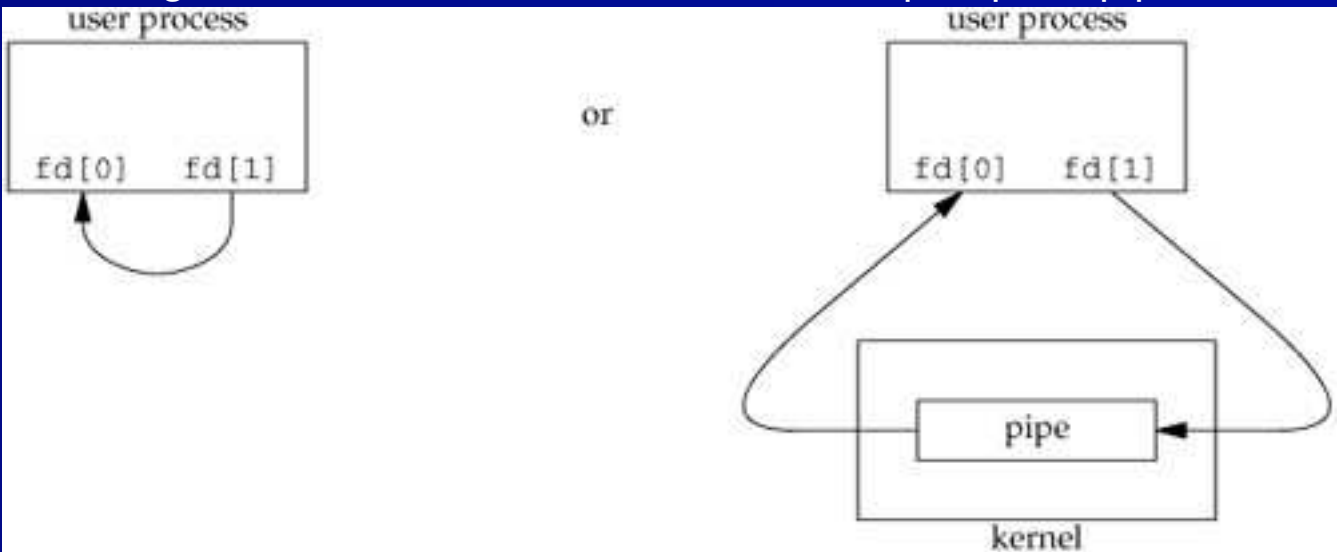


Figura 2 – un canal anonim, după apelul fork:

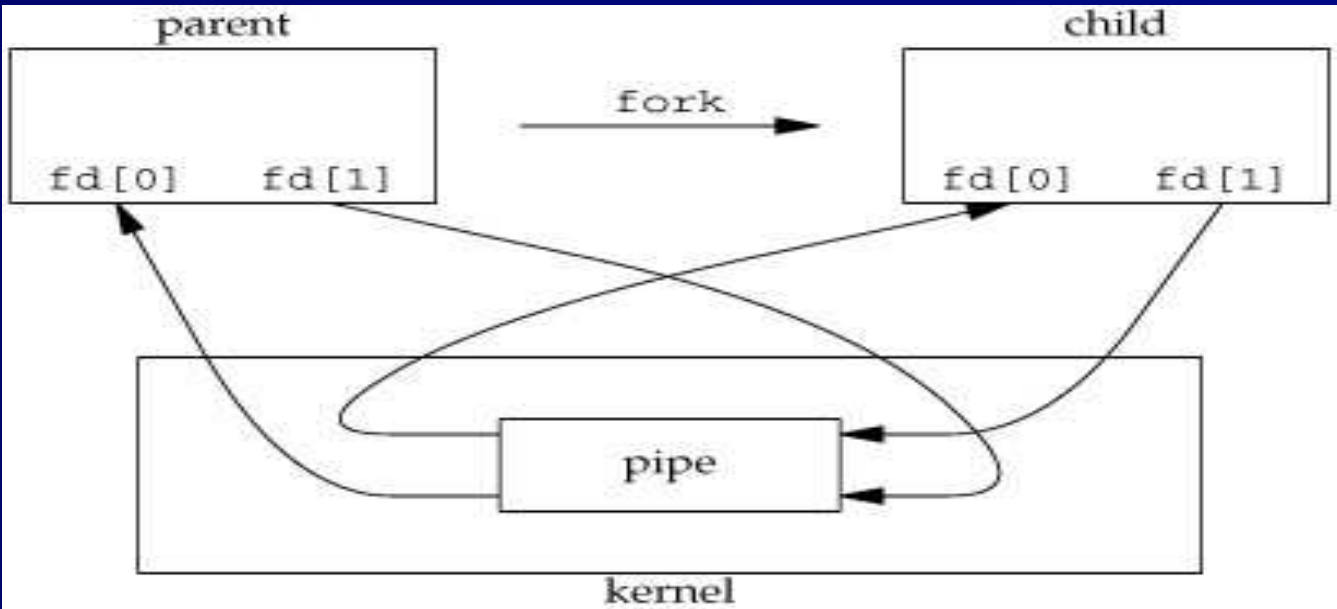
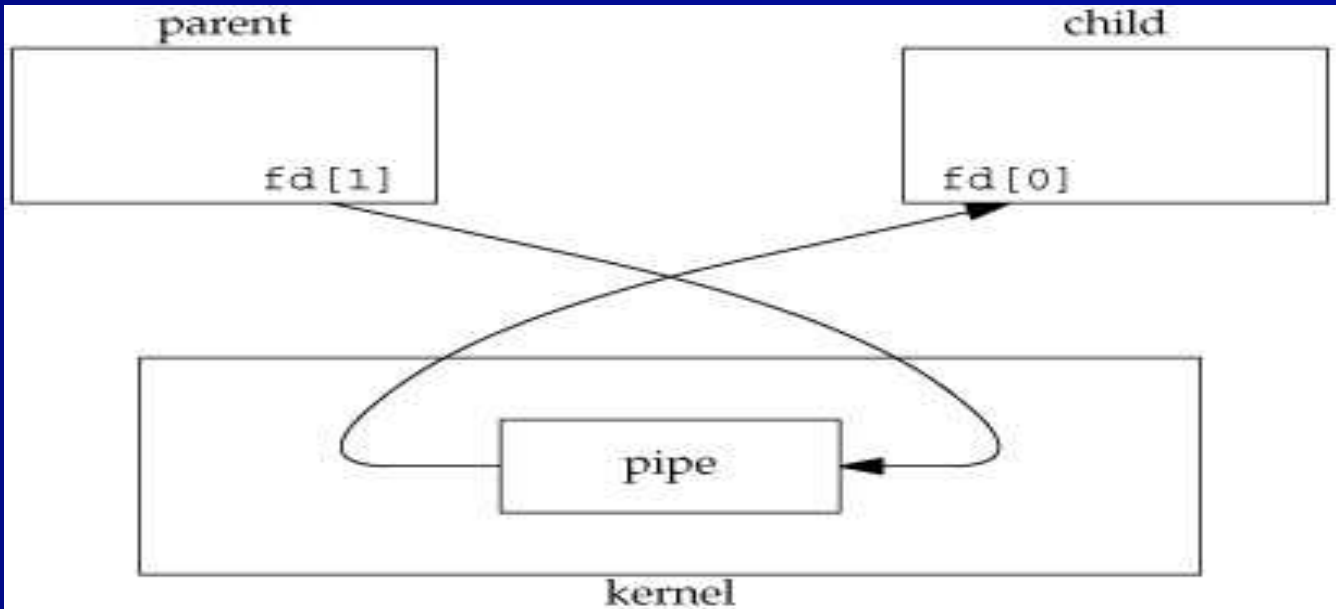


Figura 3 – un canal anonim, după configurarea “unul-la-unul”:



*Notă:* vom vedea ulterior că aceste două restricții de folosire a canalelor anonime nu mai sunt valabile și în cazul canalelor cu nume (!).

- Introducere
- Canale anonime
  - Crearea lor, cu primitiva `pipe`
  - Modul de utilizare a unui canal anonim
  - Demo: exemple de comunicație între două procese
- Canale cu nume (*fifo*)
- Caracteristici comune pentru ambele tipuri de canale
- Șabloane de comunicație între procese
- Aplicații ale canalelor de comunicație
- Referințe bibliografice



## Demo: exemple de comunicație între două procese

Introducere

Canale anonime

Crearea lor, cu primitiva `pipe`

Modul de utilizare a unui canal anonim

Demo: exemple de comunicație între două procese

Canale cu nume (*fifo*)

Caracteristici comune pentru ambele tipuri de canale

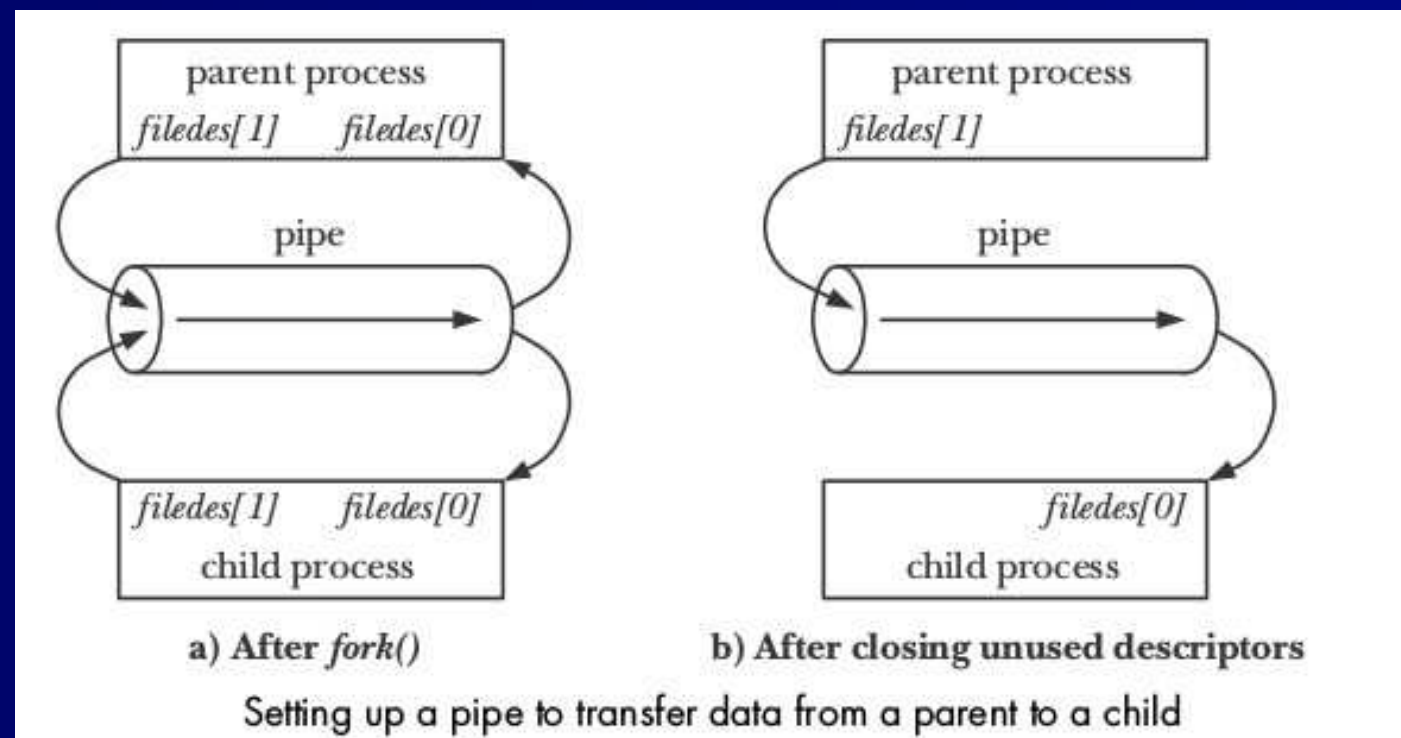
Șabloane de comunicație între procese

Aplicații ale canalelor de comunicație

Referințe bibliografice

- Exemplul #1: un program care exemplifică modul de utilizare a unui canal anonim pentru comunicația între două procese, de tipul producător–consumator. În acest exemplu se ilustrează folosirea primitivelor `read` și `write` (*i.e.*, funcțiile din API-ul POSIX) pentru a citi din canal, respectiv pentru a scrie în canal.

A se vedea fișierul sursă `pipe_ex1.c` ([2]).



Efectul acestui program: mai întâi se creează un canal anonim și un proces fiu. Apoi, procesul părinte citește o secvență de caractere de la tastatură, secvență terminată cu combinația de taste CTRL+D (*i.e.*, caracterul EOF în UNIX), și le transmite procesului fiu, prin intermediul canalului anonim, doar pe acelea care sunt litere mici. Iar procesul fiu citește din canal caracterele transmise de procesul părinte și le afișează pe ecran.

**Observație:** pentru explicații suplimentare despre acest program demonstrativ, consultați exemplul [FirstDemo – `pipe_ex1`] din suportul online de laborator ([3]).



## Demo: exemple de comunicație între două procese (cont.)

Introducere

Canale anonime

Crearea lor, cu primitiva `pipe`  
Modul de utilizare a unui canal anonim

Demo: exemple de  
comunicație între două  
procese

Canale cu nume (*fifo*)

Caracteristici comune pentru  
ambele tipuri de canale

Șabloane de comunicație între  
procese

Aplicații ale canalelor de  
comunicație

Referințe bibliografice

Pentru comunicația prin intermediul canalelor anonime se pot folosi și funcțiile I/O de nivel înalt :

- Exemplul #2: un alt program care exemplifică folosirea unui canal anonim pentru comunicația între două procese, de tipul producător–consumator.

De această dată, se utilizează funcțiile `fscanf` și, respectiv, `fprintf` (*i.e.*, din biblioteca `stdio`) pentru a citi din canal și, respectiv, pentru a scrie în canal.

A se vedea fișierul sursă `pipe_ex2.c` ([2]).

*Notă:* în acest caz, este necesară conversia descriptorilor de fișiere de la tipul `int` (*i.e.*, descriptorii folosiți de apelurile I/O din API-ul POSIX) la descriptori de tipul `FILE*` (*i.e.*, descriptorii folosiți de funcțiile I/O din biblioteca `stdio`), lucru realizabil cu ajutorul funcției de bibliotecă `fdopen`.

Efectul acestui program: mai întâi, se creează un canal anonim și un proces fiu. Apoi, procesul tată citește o secvență de numere de la tastatură, secvență terminată cu combinația de taste CTRL+D (*i.e.*, caracterul EOF în UNIX), și le transmite procesului fiu, prin intermediul canalului anonim. Iar procesul fiu citește din canal numerele trasmise de procesul părinte și le afișează pe ecran.

*Observație:* pentru explicații suplimentare despre acest program demonstrativ, consultați exemplul [SecondDemo – `pipe_ex2`] din suportul online de laborator ([3]).





# Agenda

Introducere

Canale anonime

Canale cu nume (*fifo*)

Crearea lor, cu primitiva `mkfifo`  
Modul de utilizare a unui canal cu nume  
Despre persistența informației dintr-un fișier *fifo*  
Deosebiri ale canalelor cu nume față de cele anonime

Caracteristici comune pentru ambele tipuri de canale

Șabloane de comunicație între procese

Aplicații ale canalelor de comunicație

Referințe bibliografice

Introducere

## Canale anonime

Crearea lor, cu primitiva `pipe`  
Modul de utilizare a unui canal anonim  
*Demo*: exemple de comunicație între două procese

## Canale cu nume (*fifo*)

Crearea lor, cu primitiva `mkfifo`  
Modul de utilizare a unui canal cu nume  
Despre persistența informației dintr-un fișier *fifo*  
Deosebiri ale canalelor cu nume față de cele anonime

## Caracteristici comune pentru ambele tipuri de canale

Caracteristici și restricții ale canalelor de comunicație  
Comportamentul implicit, de tip blocant  
Comportamentul de tip neblocant

## Șabloane de comunicație între procese

Clasificarea șabloanelor de comunicație inter-procese  
Șablonul de comunicație *unul-la-unul*  
Șablonul de comunicație *unul-la-multi*  
Șablonul de comunicație *multi-la-unul*  
Șablonul de comunicație *multi-la-multi*

## Aplicații ale canalelor de comunicație

Aplicația #1: implementarea unui semafor  
Aplicația #2: implementarea unei aplicații client/server

## Referințe bibliografice



## Crearea lor, cu primitiva `mkfifo`

Introducere

Canale anonime

Canale cu nume (*fifo*)

Crearea lor, cu primitiva  
`mkfifo`

Modul de utilizare a unui canal  
cu nume

Despre persistența informației  
dintr-un fișier *fifo*

Deosebiri ale canalelor cu  
nume față de cele anonime

Caracteristici comune pentru  
ambele tipuri de canale

Șabloane de comunicație între  
procese

Aplicații ale canalelor de  
comunicație

Referințe bibliografice

Un *canal cu nume* se creează cu ajutorul primitivei `mkfifo`.

Interfața acestei funcții este următoarea ([5]):

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

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

**Efect:** în urma execuției primitivei `mkfifo` se creează un canal cu nume, dar *fără a fi deschis la ambele capete* (!), precum se întâmplă în cazul creării unui canal anonim.

*Notă:* crearea unui fișier *fifo* se mai poate face cu ajutorul primitivei `mknod` apelată cu *flag*-ul `S_IFIFO`. De asemenea, mai poate fi creat și direct de la linia de comandă (*i.e.*, prompterul *shell*-ului), cu comenzile `mkfifo` sau `mknod`.

Exemplu de creare a unui fișier *fifo*: a se vedea fișierul sursă `mkfifo_ex.c` ([2]).

*Observație:* pentru explicații suplimentare despre acest program demonstrativ, consultați exemplul [ThirdDemo – `mkfifo_ex`] din suportul online de laborator ([3]).





## Modul de utilizare a unui canal cu nume

Introducere

Canale anonime

Canale cu nume (*fifo*)

Crearea lor, cu primitiva

`mkfifo`

Modul de utilizare a unui canal  
cu nume

Despre persistența informației  
dintr-un fișier *fifo*

Deosebiri ale canalelor cu  
nume față de cele anonime

Caracteristici comune pentru  
ambele tipuri de canale

Șabloane de comunicație între  
procese

Aplicații ale canalelor de  
comunicație

Referințe bibliografice

După crearea unui canal cu nume, folosirea sa pentru comunicația locală între două (sau mai multe) procese se face prin scrierea informației în acest canal și, respectiv, prin citirea informației din canal.

Iar scrierea în canal și, respectiv, citirea din canal se efectuează la fel ca pentru fișierele obișnuite. Și anume: mai întâi se deschide *explicit* fișierul la “capătul” dorit (cel de citire și/sau cel de scriere), pentru a se obține descriptorul necesar, apoi se scrie în el și/sau se citește din el, prin intermediul descriptorului obținut explicit, i.e. folosind apelurile de sistem `read` și `write`, sau cu funcțiile de citire/scriere din biblioteca `stdio`, iar la sfârșit se închide descriptorul respectiv.

\* \* \*

*Observație importantă:* deoarece acest tip de canale nu sunt *anonime* (i.e., au nume prin care pot fi referite), pot fi utilizate pentru comunicație între **orice procese care cunosc numele fișierului *fifo* respectiv**, deci nu mai avem restricția de la canale anonime, aceea că procesele trebuiau să fie “înrudite” prin `fork/exec`.



## Modul de utilizare a unui canal cu nume (cont.)

[Introducere](#)

[Canale anonime](#)

[Canale cu nume \(\*fifo\*\)](#)

[Crearea lor, cu primitiva](#)

[mkfifo](#)

[Modul de utilizare a unui canal cu nume](#)

[Despre persistența informației dintr-un fișier \*fifo\*](#)

[Deosebiri ale canalelor cu nume față de cele anonime](#)

[Caracteristici comune pentru ambele tipuri de canale](#)

[Șabloane de comunicație între procese](#)

[Aplicații ale canalelor de comunicație](#)

[Referințe bibliografice](#)

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

La fel ca pentru fișiere obișnuite, “deschiderea” unui fișier *fifo* se face explicit, printr-un apel al funcției `open` sau `fopen`, într-unul din următoarele trei moduri posibile, specificat prin parametrul transmis funcției de deschidere:

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

*Observație 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, apelul funcției de deschidere rămâne blocat (*i.e.*, funcția nu returnează) până când un alt proces va deschide celălalt capăt al canalului.



## Despre persistența informației dintr-un fișier *fifo*

Introducere

Canale anonime

Canale cu nume (*fifo*)

Crearea lor, cu primitiva

`mkfifo`

Modul de utilizare a unui canal  
cu nume

**Despre persistența informației  
dintr-un fișier *fifo***

Deosebiri ale canalelor cu  
nume față de cele anonime

Caracteristici comune pentru  
ambele tipuri de canale

Șabloane de comunicație între  
procese

Aplicații ale canalelor de  
comunicație

Referințe bibliografice

*Observație:* un *canal cu nume* este creat tot în memoria internă a sistemului (ca și unul anonim), dar în plus are asociat un nume, reprezentat printr-un fișier de tipul special *fifo*, care este păstrat în sistemul de fișiere.

*Concluzie:* informațiile conținute în acest tip de fișiere sunt stocate în memoria principală, nu pe disc, și ca urmare nu sunt persistente. (Practic, conținutul unui fișier *fifo* este gestionat, de către nucleul SO-ului, tot ca o coadă FIFO aflată în memorie, la fel ca și în cazul canalelor anonime.)

Așadar, **perioada de retenție** a informației stocate într-un canal este următoarea: Spre deosebire de fișierele obișnuite ( ce păstrează informația scrisă în ele pe o perioadă nedeterminată – mai precis, până la o eventuală operație de modificare sau ștergere ), în cazul unui fișier *fifo* informația scrisă în el se păstrează doar din momentul scrierii și până în momentul când atât procesul care a scris acea informație, cât și orice alt proces ce-l accesa, termină accesul la acel canal *fifo* (închizându-și capetele canalului), iar aceasta numai dacă informația nu este consumată mai devreme, prin citire.

*Demo:* a se vedea fișierul sursă `testare_retentie_fifo.c` ([2]).

*Notă:* pentru mai multe explicații despre acest program și descrierea comportamentului său la execuție, consultați exemplul `[FourthDemo – testare_retentie_fifo]` din suportul online de laborator ([3]).





## Deosebiri ale canalelor cu nume față de cele anonime

[Introducere](#)

[Canale anonime](#)

[Canale cu nume \(\*fifo\*\)](#)

[Crearea lor, cu primitiva](#)

[mkfifo](#)

[Modul de utilizare a unui canal  
cu nume](#)

[Despre persistența informației  
dintr-un fișier \*fifo\*](#)

[Deosebiri ale canalelor cu  
nume față de cele anonime](#)

[Caracteristici comune pentru  
ambele tipuri de canale](#)

[Șabloane de comunicație între  
procese](#)

[Aplicații ale canalelor de  
comunicație](#)

[Referințe bibliografice](#)

- Funcția de creare a unui fișier *fifo* (*i.e.*, canal cu nume) nu realizează și deschiderea automată a celor două capete ale canalului, precum la canalele anonime, ci acestea trebuie să fie deschise explicit, după creare, prin apelul unei funcții de deschidere a acelui fișier.
- Un canal *fifo* poate fi deschis, la oricare dintre capete, de orice proces, indiferent dacă acel proces are sau nu vreo legătură de “rudenie” (prin `fork/exec`) cu procesul care a creat canalul respectiv.  
Aceasta este posibil deoarece un proces trebuie doar să cunoască numele fișierului *fifo* pe care dorește să-l deschidă, pentru a-l putea deschide. Evident, mai trebuie și ca procesul respectiv să aibă drepturi de acces pentru acel fișier *fifo*.
- După ce un proces închide un capăt al unui canal *fifo*, acel proces poate redeschide din nou acel capăt al canalului.  
Motivul pentru care ar dori aceasta: poate constata, ulterior momentului închiderii, că are nevoie să mai efectueze și alte operații I/O asupra acelui capăt.



# Agenda

Introducere

Canale anonime

Canale cu nume (*fifo*)

Caracteristici comune pentru ambele tipuri de canale

Caracteristici și restricții ale canalelor de comunicație

Comportamentul implicit, de tip blocant

Comportamentul de tip neblocant

Șabloane de comunicație între procese

Aplicații ale canalelor de comunicație

Referințe bibliografice

Introducere

## Canale anonime

Crearea lor, cu primitiva `pipe`

Modul de utilizare a unui canal anonim

*Demo*: exemple de comunicație între două procese

## Canale cu nume (*fifo*)

Crearea lor, cu primitiva `mkfifo`

Modul de utilizare a unui canal cu nume

Despre persistența informației dintr-un fișier *fifo*

Deosebiri ale canalelor cu nume față de cele anonime

## Caracteristici comune pentru ambele tipuri de canale

Caracteristici și restricții ale canalelor de comunicație

Comportamentul implicit, de tip blocant

Comportamentul de tip neblocant

## Șabloane de comunicație între procese

Clasificarea șabloanelor de comunicație inter-procese

Șablonul de comunicație *unul-la-unul*

Șablonul de comunicație *unul-la-multi*

Șablonul de comunicație *multi-la-unul*

Șablonul de comunicație *multi-la-multi*

## Aplicații ale canalelor de comunicație

Aplicația #1: implementarea unui semafor

Aplicația #2: implementarea unei aplicații client/server

## Referințe bibliografice





# Caracteristici și restricții ale canalelor de comunicație

Introducere

Canale anonime

Canale cu nume (*fifo*)

Caracteristici comune pentru  
ambele tipuri de canale

Caracteristici și restricții ale  
canalelor de comunicație

Comportamentul implicit, de tip  
blocant

Comportamentul de tip  
neblocant

Șabloane de comunicație între  
procese

Aplicații ale canalelor de  
comunicație

Referințe bibliografice

- Ambele tipuri de canale sunt canale *unidirecționale*, adică pe la un capăt se scrie informația în canal, iar pe la capătul opus se citește.  
*Notă:* însă putem avea mai mulți scriitori (*i.e.*, toate procesele ce au acces la capătul de scriere, pot să scrie în canal), și/sau mai multi cititori (*i.e.*, toate procesele ce au acces la capătul de citire, pot să citească din canal).
- Unitatea de informație pentru ambele tipuri de canale este *octetul*.  
Cu alte cuvinte, cantitatea minimă de informație ce poate fi scrisă în canal, respectiv citită din canal, este de 1 octet.
- Capacitatea unui canal de comunicație este limitată la o anumită dimensiune maximă (*e.g.*, 4 Ko, 16 Ko, 64 Ko, ș.a.), ce este configurabilă.  
Spre exemplu, în Linux (începând de la versiunea 2.6.35) se poate afla, respectiv configura, capacitatea unui canal de comunicație prin operațiile `F_GETPIPE_SZ`, respectiv `F_SETPIPE_SZ`, disponibile prin apelul de sistem `fcntl`.  
Pentru detalii, consultați documentația acestui apel (*i.e.*, `man 2 fcntl`), precum și explicațiile prezentate în exercițiul rezolvat [\[A pipe's capacity\]](#).



## Caracteristici și restricții ale canalelor de comunicație (cont.)

[Introducere](#)

[Canale anonime](#)

[Canale cu nume \(\*fifo\*\)](#)

[Caracteristici comune pentru ambele tipuri de canale](#)

[Caracteristici și restricții ale canalelor de comunicație](#)

[Comportamentul implicit, de tip blocant](#)

[Comportamentul de tip neblocant](#)

[Șabloane de comunicație între procese](#)

[Aplicații ale canalelor de comunicație](#)

[Referințe bibliografice](#)

- Practic, ambele tipuri de canale (*i.e.*, și cele anonime, și cele cu nume) funcționează 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*) informației citite (!), iar scrierea în canal se face prin “inserarea” în coadă a informației scrise.

*Concluzie:* așadar, citirea dintr-un fișier *fifo* diferă de citirea din fișiere obișnuite, pentru care citirea se face fără consumarea informației din fișier.

- În cazul fișierelor obișnuite am văzut că există noțiunea de *offset* (*i.e.*, poziția curentă în fișier, de la care se efectuează operația curentă de citire sau scriere). În schimb, nici pentru fișierele *fifo*, nici pentru canalele anonime nu există această noțiune de *offset*, ele funcționând precum o coadă FIFO.



# Comportamentul implicit, de tip blocant

[Introducere](#)

[Canale anonime](#)

[Canale cu nume \(fifo\)](#)

[Caracteristici comune pentru ambele tipuri de canale](#)

[Caracteristici și restricții ale canalelor de comunicație](#)

[Comportamentul implicit, de tip blocant](#)

[Comportamentul de tip neblocant](#)

[Șabloane de comunicație între procese](#)

[Aplicații ale canalelor de comunicație](#)

[Referințe bibliografice](#)

- Citirea dintr-un canal de comunicație funcționează în felul următor:
  - Apelul de citire `read` va citi din canal și va returna imediat, fără să se blocheze, numai dacă mai este suficientă informație în canal, iar în acest caz valoarea returnată reprezintă numărul de octeți citați din canal.
  - Altfel, dacă canalul este gol, sau nu conține suficientă informație, apelul de citire `read` va rămâne blocat până când va avea suficientă informație în canal pentru a putea citi cantitatea de informație specificată, ceea ce se va întâmpla în momentul când un alt proces va scrie în canal.
  - Alt caz de excepție la citire: dacă un proces încearcă să citească din canal și niciun proces nu mai este capabil să scrie în canal (deoarece toate procesele și-au închis deja capătul de scriere), atunci apelul `read` returnează imediat valoarea 0 prin care se semnalizează că “a citit EOF” din canal.  
*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).





## Comportamentul implicit, de tip blocant (cont.)

Introducere

Canale anonime

Canale cu nume (*fifo*)

Caracteristici comune pentru  
ambele tipuri de canale

Caracteristici și restricții ale  
canalelor de comunicație

Comportamentul implicit, de tip  
blocant

Comportamentul de tip  
neblocant

Șabloane de comunicație între  
procese

Aplicații ale canalelor de  
comunicație

Referințe bibliografice

- Scrierea într-un canal de comunicație funcționează în felul următor:
  - Apelul de scriere `write` va scrie în canal și va returna imediat, fără să se blocheze, numai dacă mai este suficient spațiu liber în canal, iar în acest caz valoarea returnată reprezintă numărul de octeți efectiv scriși în canal (care poate să nu coincidă întotdeauna cu numărul de octeți ce se doreau a se scrie, căci pot apare eventuale erori I/O).
  - Altfel, dacă canalul este plin, sau nu conține suficient spațiu liber, apelul de scriere `write` va rămâne blocat până când va avea suficient spațiu liber în canal pentru a putea scrie informația specificată ca argument, ceea ce se va întâmpla în momentul când un alt proces va citi din canal.
  - Alt caz de excepție la scriere: dacă un proces încearcă să scrie în canal și niciun proces nu mai este capabil să citească din canal (deoarece toate procesele și-au închis deja capătul de citire), atunci sistemul va trimite acelui proces *semnalul* **SIGPIPE**, ce cauzează terminarea forțată a procesului, fără a afișa însă vreun mesaj de eroare (Notă: versiunile mai vechi de Linux afișau “**Broken pipe**”).



## Comportamentul implicit, de tip blocant (cont.)

[Introducere](#)

[Canale anonime](#)

[Canale cu nume \(\*fifo\*\)](#)

[Caracteristici comune pentru ambele tipuri de canale](#)

[Caracteristici și restricții ale canalelor de comunicație](#)

[Comportamentul implicit, de tip blocant](#)

[Comportamentul de tip neblocant](#)

[Șabloane de comunicație între procese](#)

[Aplicații ale canalelor de comunicație](#)

[Referințe bibliografice](#)

*Observație:* în locul primitivelor `read` și `write` din API-ul POSIX, putem folosi funcțiile I/O de nivel înalt din biblioteca `stdio` pentru a citi din canal (e.g., cu `fread`, `fscanf`, ș.a.) și, respectiv, pentru a scrie în canal (e.g., cu `fwrite`, `fprintf`, ș.a.). Și aceste funcții de bibliotecă au un *comportament implicit blocant*, similar cu cel descris mai sus, singura diferență fiind aceea că, reamintiți-vă, aceste funcții lucrează *buffer-izat* (!), i.e. folosind un *cache* local în *user-space*.

*Consecință:* modul de lucru *buffer-izat* al funcțiilor I/O din `stdio`, poate cauza uneori erori logice (i.e., *bug-uri*) dificil de depistat, datorate neatenției programatorului, care poate uita să forțeze “golirea” *buffer*-ului în canal cu ajutorul funcției `fflush`, imediat după apelul funcției de scriere utilizate pentru a scrie acea informație în canal.

Și astfel, un proces cititor al acelei informații va rămâne blocat în apelul de citire, deoarece informația încă nu a ajuns în canal, iar programatorul va căuta cauza blocajului în altă parte, crezând că informația, pe care o scrisese, a ajuns “instantaneu” (i.e., *fără nicio întârziere sesizabilă*) în canal (!).

*Recomandare:* acordați mare atenție să nu comiteți acest gen de greșeli logice, căci le-am observat de nenumărate ori, pe parcursul anilor, în programele scrise de studenți.





# Comportamentul de tip neblokant

[Introducere](#)

[Canale anonime](#)

[Canale cu nume \(\*fifo\*\)](#)

[Caracteristici comune pentru ambele tipuri de canale](#)

[Caracteristici și restricții ale canalelor de comunicație](#)

[Comportamentul implicit, de tip blokant](#)

[Comportamentul de tip neblokant](#)

[Șabloane de comunicație între procese](#)

[Aplicații ale canalelor de comunicație](#)

[Referințe bibliografice](#)

Cele descrise mai devreme, despre blocarea apelurilor de citire, respectiv de scriere, în cazul canalului gol, respectiv plin, corespund comportamentului implicit, de tip ***blokant***, al canalelor de comunicație.

Acest comportament implicit poate fi modificat, pentru ambele tipuri de canale de comunicație, într-un comportament de tip ***neblokant***, situație în care apelurile de citire și, respectiv, de scriere, nu mai rămân blocate în cazul canalului gol și, respectiv, în cazul canalului plin, ci returnează imediat valoarea `-1`, setând în mod corespunzător variabila `errno`.

Mai mult, putem modifica *separat* comportamentul pentru oricare dintre cele două capete ale unui canal, nu suntem limitați doar la a schimba *simultan* comportamentul pentru ambele capete (!).

În plus, în cazul canalelor cu nume, o deschidere *neblokantă* a unuia dintre capetele canalului va reuși imediat, fără să mai aștepte ca vreun alt proces să deschidă celălalt capăt, precum se întâmplă în cazul deschiderii implicite, de tip blokant.



## Comportamentul de tip neblokant (cont.)

[Introducere](#)

[Canale anonime](#)

[Canale cu nume \(\*fifo\*\)](#)

[Caracteristici comune pentru ambele tipuri de canale](#)

[Caracteristici și restricții ale canalelor de comunicație](#)

[Comportamentul implicit, de tip blokant](#)

[Comportamentul de tip neblokant](#)

[Șabloane de comunicație între procese](#)

[Aplicații ale canalelor de comunicație](#)

[Referințe bibliografice](#)

Modificarea comportamentului implicit în comportament ***neblokant*** se realizează prin setarea atributului **`O_NONBLOCK`** pentru descriptorul corespunzător aceluși capăt al canalului de comunicație pentru care se dorește modificarea comportamentului.

Setarea atributului **`O_NONBLOCK`** pentru descriptorul dorit, se poate face astfel:

1. fie direct la deschiderea explicită a canalului, *e.g.* printr-un apel de forma:  
**`fd_out = open("canal_fifo", O_WRONLY | O_NONBLOCK) ;`**  
care va seta la deschidere atributul **`O_NONBLOCK`** doar pentru capătul de scriere.  
Această modalitate este posibilă numai pentru canale cu nume (*i.e.*, fișiere *fifo*).
2. fie după deschiderea, implicită sau explicită, a canalului, utilizând primitiva **`fcntl`**, *e.g.* printr-un apel de forma: **`fcntl(fd_out, F_SETFL, O_NONBLOCK) ;`**  
Această modalitate este posibilă pentru ambele tipuri de canale.

***Exercițiu:*** scrieți un program prin care să determinați capacitatea ambelor tipuri de canale de comunicație pe sistemul Linux pe care lucrați.

***Rezolvare:*** dacă nu reușiți să-l rezolvați singuri, citiți exercițiile rezolvate [\[A pipe's capacity\]](#) și [\[A fifo's capacity\]](#) prezentate în [suportul de laborator](#).



# Agenda

Introducere

Canale anonime

Canale cu nume (*fifo*)

Caracteristici comune pentru  
ambele tipuri de canale

Șabloane de comunicație între  
procese

Clasificarea șabloanelor de  
comunicație inter-procese

Șablonul de comunicație  
*unul-la-unul*

Șablonul de comunicație  
*unul-la-multi*

Șablonul de comunicație  
*multi-la-unul*

Șablonul de comunicație  
*multi-la-multi*

Aplicații ale canalelor de  
comunicație

Referințe bibliografice

Introducere

## Canale anonime

- Crearea lor, cu primitiva `pipe`
- Modul de utilizare a unui canal anonim
- Demo*: exemple de comunicație între două procese

## Canale cu nume (*fifo*)

- Crearea lor, cu primitiva `mkfifo`
- Modul de utilizare a unui canal cu nume
- Despre persistența informației dintr-un fișier *fifo*
- Deosebiri ale canalelor cu nume față de cele anonime

## Caracteristici comune pentru ambele tipuri de canale

- Caracteristici și restricții ale canalelor de comunicație
- Comportamentul implicit, de tip blocant
- Comportamentul de tip neblocant

## Șabloane de comunicație între procese

- Clasificarea șabloanelor de comunicație inter-procese
- Șablonul de comunicație *unul-la-unul*
- Șablonul de comunicație *unul-la-multi*
- Șablonul de comunicație *multi-la-unul*
- Șablonul de comunicație *multi-la-multi*

## Aplicații ale canalelor de comunicație

- Aplicația #1: implementarea unui semafor
- Aplicația #2: implementarea unei aplicații client/server

## Referințe bibliografice



# Clasificarea șabloanelor de comunicație inter-procese

[Introducere](#)

[Canale anonime](#)

[Canale cu nume \(\*fifo\*\)](#)

[Caracteristici comune pentru ambele tipuri de canale](#)

[Șabloane de comunicație între procese](#)

[Clasificarea șabloanelor de comunicație inter-procese](#)

[Șablonul de comunicație unul-la-unul](#)

[Șablonul de comunicație unul-la-multi](#)

[Șablonul de comunicație multi-la-unul](#)

[Șablonul de comunicație multi-la-multi](#)

[Aplicații ale canalelor de comunicație](#)

[Referințe bibliografice](#)

După numărul de procese “scriitori” și, respectiv, de procese “cititori” ce utilizează un anumit canal de comunicație (anonim sau cu nume) pentru a comunica între ele, putem diferenția următoarele șabloane de comunicație inter-procese:

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





# Șablonul de comunicație *unul-la-unul*

Introducere

Canale anonime

Canale cu nume (*fifo*)

Caracteristici comune pentru  
ambele tipuri de canale

Șabloane de comunicație între  
procese

Clasificarea șabloanelor de  
comunicație inter-procese

Șablonul de comunicație  
*unul-la-unul*

Șablonul de comunicație  
*unul-la-multi*

Șablonul de comunicație  
*multi-la-unul*

Șablonul de comunicație  
*multi-la-multi*

Aplicații ale canalelor de  
comunicație

Referințe bibliografice

Comunicația *unul-la-unul* reprezintă șablonul cel mai simplu, neridicând probleme deosebite de implementare. Din acest motiv, este și cel mai folosit în practică.

*Exemple:* cele două programe demonstrative prezentate anterior, în secțiunea despre canale anonime, se încadrează în acest șablon de comunicație.

*Demo:* exercițiile rezolvate [**Producer-consumer** pattern #1, (v2, using fifos for IPC)] și, respectiv, [**Producer-consumer** pattern #2, (v2, using fifos for IPC)] din suportul online de laborator ([3]), ilustrează alte două programe care, fiecare în parte, utilizează un canal cu nume pentru comunicația *unul-la-unul* între două procese, unul cu rol de producător, iar celălalt cu rol de consumator.

\* \* \*

Celelalte trei șabloane ridică anumite probleme de sincronizare, datorate accesului concurent al mai multor procese la câte unul, sau la ambele, dintre capetele canalului, probleme de care trebuie să se țină cont la implementarea acestor șabloane.

Vom trece în revistă, pe rând, aceste probleme de sincronizare, ce pot avea efecte asupra *integrității datelor* transmise prin canal (*i.e.*, “coruperea” mesajelor).





# Șablonul de comunicație *unul-la-mulți*

Introducere

Canale anonime

Canale cu nume (*fifo*)

Caracteristici comune pentru  
ambele tipuri de canale

Șabloane de comunicație între  
procese

Clasificarea șabloanelor de  
comunicație inter-procese

Șablonul de comunicație  
*unul-la-unul*

Șablonul de comunicație  
*unul-la-mulți*

Șablonul de comunicație  
*mulți-la-unul*

Șablonul de comunicație  
*mulți-la-mulți*

Aplicații ale canalelor de  
comunicație

Referințe bibliografice

Factori ce pot genera anumite probleme de sincronizare, cu efecte asupra *integrității datelor*:

## ■ *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*

Pot apare probleme de sincronizare, deoarece mesajele nu mai pot fi citite *atomic*. Soluția este folosirea mesajelor formate astfel:

MESAJ = HEADER + MESAJUL PROPRIU-ZIS ,

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

*Protocolul de comunicație* utilizat: sunt necesare 2 apeluri `read` pentru a citi un mesaj în întregime, de aceea trebuie garantat accesul exclusiv la canal (folosind, spre exemplu, blocaje pe fișiere).



## Șablonul de comunicație *unul-la-mulți* (cont.)

Introducere

Canale anonime

Canale cu nume (*fifo*)

Caracteristici comune pentru  
ambele tipuri de canale

Șabloane de comunicație între  
procese

Clasificarea șabloanelor de  
comunicație inter-procese

Șablonul de comunicație  
*unul-la-unul*

Șablonul de comunicație  
*unul-la-mulți*

Șablonul de comunicație  
*mulți-la-unul*

Șablonul de comunicație  
*mulți-la-mulți*

Aplicații ale canalelor de  
comunicație

Referințe bibliografice

Factori ce pot genera anumite probleme de sincronizare, cu efecte asupra *integrității datelor*:

### ■ *destinatarul* mesajelor:

#### — *mesaje cu destinatar oarecare*

Nu ridică probleme deosebite – fiecare mesaj poate fi citit și prelucrat de oricare dintre procesele “cititori”.

#### — *mesaje cu destinatar specificat*

Trebuie asigurat faptul că mesajul este citit exact de către “cititorul” căruia îi era destinat. Soluția – am putea folosi mesaje formate astfel:

MESAJ = HEADER + MESAJUL PROPRIU-ZIS ,

*header*-ul conținând un identificator al destinatarului.

Pentru citire, s-ar putea aplica protocolul de comunicație discutat la mesaje de lungime variabilă. Însă, apare o *problemă suplimentară*: dacă un “cititor” a citit un mesaj care nu-i era destinat lui, cum facem să-l livrăm celui căruia îi era destinat? O soluție ar fi să îl scrie înapoi în canal, și apoi va face o pauză aleatoare înainte de a încerca să citească din nou din canal.

*Notă*: această soluție poate suferi de fenomenul de *starvation*.



# Șablonul de comunicație *mulți-la-unul*

Introducere

Canale anonime

Canale cu nume (*fifo*)

Caracteristici comune pentru  
ambele tipuri de canale

Șabloane de comunicație între  
procese

Clasificarea șabloanelor de  
comunicație inter-procese

Șablonul de comunicație  
*unul-la-unul*

Șablonul de comunicație  
*unul-la-mult*

Șablonul de comunicație  
*mulți-la-unul*

Șablonul de comunicație  
*mulți-la-mult*

Aplicații ale canalelor de  
comunicație

Referințe bibliografice

Factori ce pot genera anumite probleme de sincronizare, cu efecte asupra *integrității datelor*:

## ■ *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*

Trebuie indicată “cititorului” lungimea fiecărui mesaj. Soluția este folosirea mesajelor formate astfel:

MESAJ = HEADER + MESAJUL PROPRIU-ZIS ,

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

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



## Șablonul de comunicație *mulți-la-unul* (cont.)

Introducere

Canale anonime

Canale cu nume (*fifo*)

Caracteristici comune pentru  
ambele tipuri de canale

Șabloane de comunicație între  
procese

Clasificarea șabloanelor de  
comunicație inter-procese

Șablonul de comunicație  
*unul-la-unul*

Șablonul de comunicație  
*unul-la-mult*

Șablonul de comunicație  
*mulți-la-unul*

Șablonul de comunicație  
*mulți-la-mult*

Aplicații ale canalelor de  
comunicație

Referințe bibliografice

Factori ce pot genera anumite probleme de sincronizare, cu efecte asupra *integrității datelor*:

### ■ *destinatarul* mesajelor:

#### — *mesaje cu expeditor oarecare*

Nu ridică probleme deosebite – fiecare mesaj poate fi citit de procesul “cititor” și prelucrat în același fel, indiferent de la care dintre procesele “scriitori” provine acel mesaj.

#### — *mesaje cu expeditor specificat*

Trebuie asigurat că mesajul îi indică “cititorului” care este “scriitorul” care i l-a trimis. Soluția – mesaje formate în felul următor:

MESAJ = HEADER + MESAJUL PROPRIU-ZIS ,

*header*-ul conținând 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.





# Șablonul de comunicație *mulți-la-mulți*

Introducere

Canale anonime

Canale cu nume (*fifo*)

Caracteristici comune pentru  
ambele tipuri de canale

Șabloane de comunicație între  
procese

Clasificarea șabloanelor de  
comunicație inter-procese

Șablonul de comunicație  
*unul-la-unul*

Șablonul de comunicație  
*unul-la-mulți*

Șablonul de comunicație  
*mulți-la-unul*

Șablonul de comunicație  
*mulți-la-mulți*

Aplicații ale canalelor de  
comunicație

Referințe bibliografice

Problemele de sincronizare ce pot apare în cazul acestui șablon, pot fi cauzate de oricare dintre factorii discutați la șabloanele *unul-la-mulți* și *mulți-la-unul*:

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

Tratarea acestora se poate face prin combinarea soluțiilor prezentate la șabloanele precedente.

**Notă:** pentru simplitatea programării, uneori se poate prefera înlocuirea unui singur canal folosit pentru comunicație *unul-la-mulți*, cu mai multe canale folosite pentru comunicație *unul-la-unul*, *i.e.* cu câte un canal pentru fiecare proces “cititor” existent. Evident, se poate proceda similar și pentru șabloanele *unul-la-mulți* și *mulți-la-mulți*.

**Demo:** a se vedea programele `suma_pipes.c` și `suma_fifos.c` ([2]), care reprezintă rescrieri ale programului `suma_files.c` din lecția practică despre `fork` și `wait`, prin înlocuirea fișierelor obișnuite cu canale (anonime și, respectiv, cu nume) pentru comunicațiile dintre supervisor și workeri.

Comunicațiile dinspre workeri spre supervisor folosesc șablonul *mulți-la-unul*. În schimb, șablonul *unul-la-mulți* pentru comunicațiile dinspre supervisor spre workeri l-am implementat pe baza ideii de mai sus. ( Pentru explicații suplimentare despre aceste două programe, puteți consulta exemplele [FifthDemo – `suma_pipes`] și [SixthDemo – `suma_fifos`] din suportul online de laborator ([3]). )



# Agenda

Introducere

Canale anonime

Canale cu nume (*fifo*)

Caracteristici comune pentru  
ambele tipuri de canale

Șabloane de comunicație între  
procese

Aplicații ale canalelor de  
comunicație

Aplicația #1: implementarea  
unui semafor  
Aplicația #2: implementarea  
unei aplicații client/server

Referințe bibliografice

Introducere

## Canale anonime

Crearea lor, cu primitiva `pipe`  
Modul de utilizare a unui canal anonim  
*Demo*: exemple de comunicație între două procese

## Canale cu nume (*fifo*)

Crearea lor, cu primitiva `mkfifo`  
Modul de utilizare a unui canal cu nume  
Despre persistența informației dintr-un fișier *fifo*  
Deosebiri ale canalelor cu nume față de cele anonime

## Caracteristici comune pentru ambele tipuri de canale

Caracteristici și restricții ale canalelor de comunicație  
Comportamentul implicit, de tip blocant  
Comportamentul de tip neblocant

## Șabloane de comunicație între procese

Clasificarea șabloanelor de comunicație inter-procese  
Șablonul de comunicație *unul-la-unul*  
Șablonul de comunicație *unul-la-multi*  
Șablonul de comunicație *multi-la-unul*  
Șablonul de comunicație *multi-la-multi*

## Aplicații ale canalelor de comunicație

Aplicația #1: implementarea unui semafor  
Aplicația #2: implementarea unei aplicații client/server

## Referințe bibliografice



## Aplicația #1: implementarea unui semafor

[Introducere](#)

[Canale anonime](#)

[Canale cu nume \(\*fifo\*\)](#)

[Caracteristici comune pentru  
ambele tipuri de canale](#)

[Șabloane de comunicație între  
procese](#)

[Aplicații ale canalelor de  
comunicație](#)

**[Aplicația #1: implementarea  
unui semafor](#)**

[Aplicația #2: implementarea  
unei aplicații client/server](#)

[Referințe bibliografice](#)

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

O posibilă implementare ar consta în următoarele idei:

**Inițializarea semaforului** s-ar realiza prin crearea unui fișier *fifo* de către un proces cu rol de *supervizor* (acesta poate fi oricare dintre procesele ce vor folosi acel semafor, sau poate fi un proces separat).

Acest proces *supervizor* va scrie inițial în canal 1 octet oarecare, dacă e vorba de un semafor binar (sau  $n$  octeți oarecare, dacă e vorba de un semafor general  $n$ -ar). Iar apoi va păstra deschise ambele capete ale canalului pe toată durata de execuție a proceselor ce vor folosi acel semafor (cu scopul de a nu se pierde pe parcurs informația din canal, datorită inexistenței la un moment dat a măcar unui proces care să aibă deschis măcar vreunul dintre capete, conform celor discutate anterior legat de perioada de retenție a informației într-un canal *fifo*).



## Aplicația #1: implementarea unui semafor (cont.)

[Introducere](#)

[Canale anonime](#)

[Canale cu nume \(\*fifo\*\)](#)

[Caracteristici comune pentru  
ambele tipuri de canale](#)

[Șabloane de comunicație între  
procese](#)

[Aplicații ale canalelor de  
comunicație](#)

[Aplicația #1: implementarea  
unui semafor](#)

[Aplicația #2: implementarea  
unei aplicații client/server](#)

[Referințe bibliografice](#)

**Operația `wait` va consta în citirea unui octet din fișierul *fifo*.**

Mai precis, întâi se va face deschiderea lui, urmată de citirea efectivă a unui octet, și apoi eventual închiderea fișierului.

**Operația `signal` va consta în scrierea unui octet în fișierul *fifo*.**

Mai precis, întâi se va face deschiderea lui, urmată de scrierea efectivă a unui octet, și apoi eventual închiderea fișierului.

**Observații:**

i) citirea se va face, în modul implicit, *blocant*, ceea ce va asigura așteptarea procesului la punctul de intrare în secțiunea sa critică atunci când semaforul este “pe roșu”, adică dacă canalul *fifo* este gol.

ii) scrierea nu se va putea bloca (cu condiția ca  $n$ -ul semaforului general să nu depășească capacitatea maximă pe care o putem configura pentru un canal).

*Temă:* implementați în C un semafor binar pe baza ideilor de mai sus și scrieți un program demonstrativ în care să utilizați semaforul astfel implementat pentru asigurarea excluderii mutuale a unei secțiuni critice de cod (pentru “inspirație” în scrierea programului demonstrativ, revedeți problemele de sincronizare discutate în cursurile teoretice #5 și #6).





## Aplicația #2: implementarea unei aplicații client/server

Introducere

Canale anonime

Canale cu nume (*fifo*)

Caracteristici comune pentru  
ambele tipuri de canale

Șabloane de comunicație între  
procese

Aplicații ale canalelor de  
comunicație

Aplicația #1: implementarea  
unui semafor

Aplicația #2: implementarea  
unei aplicații client/server

Referințe bibliografice

O *aplicație cu arhitectură de tip client/server* este compusă din două componente:

- **serverul**: este un program care dispune de un anumit număr de *servicii* (*i.e.*, funcții, operații, etc.), pe care le pune la dispoziția clienților.
- **clientul**: este un program care “interoghează” serverul, solicitându-i *efectuarea unui serviciu* (dintre cele puse la dispoziție de acel server).

*Exemplu*: Browserele pe care le folosiți 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 afișată în fereastra grafică a *browserului*.

*Implementarea unei aplicații de tip client/server* o puteți face în felul următor:

Programul server va fi rulat în *background*, și va sta în așteptarea cererilor din partea clienților, putând servi mai mulți clienți simultan.

Iar clienții vor putea fi rulați mai mulți simultan (din același cont și/sau din conturi utilizator diferite), și se vor “conecta” la serverul rulat în *background*.

*Notă*: pot exista, la un moment dat, mai multe procese client care încearcă, fiecare independent de celelalte, să folosească serviciile puse la dispoziție de același proces server.



## Aplicația #2: implementarea unei aplicații client/server (cont.)

[Introducere](#)

[Canale anonime](#)

[Canale cu nume \(fifo\)](#)

[Caracteristici comune pentru ambele tipuri de canale](#)

[Șabloane de comunicație între procese](#)

[Aplicații ale canalelor de comunicație](#)

[Aplicația #1: implementarea unui semafor](#)

[Aplicația #2: implementarea unei aplicații client/server](#)

[Referințe bibliografice](#)

*Observație:* În realitate, programul server este rulat pe un anumit calculator, iar clienții pe diverse alte calculatoare, conectate la INTERNET, comunicația realizându-se folosind *socket*-uri, prin intermediul rețelelor de calculatoare.

Însă puteți simula această “realitate” folosind **comunicație prin canale cu nume și executând toate procesele (i.e., serverul și clienții) pe un același calculator**, eventual din conturi utilizator diferite.

Tipurile de servere existente în realitate, d.p.d.v. al *servirii “simultane” a mai multor clienți*, 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 clienți. Deci nu poate rezolva mai mulți clienți în același 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 clienți. Deci poate rezolva mai mulți clienți în același timp!



## Aplicația #2: implementarea unei aplicații client/server (cont.)

[Introducere](#)

[Canale anonime](#)

[Canale cu nume \(\*fifo\*\)](#)

[Caracteristici comune pentru ambele tipuri de canale](#)

[Șabloane de comunicație între procese](#)

[Aplicații ale canalelor de comunicație](#)

[Aplicația #1: implementarea unui semafor](#)

[Aplicația #2: implementarea unei aplicații client/server](#)

[Referințe bibliografice](#)

### Detalii legate de implementare:

- Pentru implementarea unui server de tip iterativ este suficient un singur proces secvențial. În schimb, pentru implementarea unui server de tip concurent este nevoie de mai multe procese secvențiale: un proces *supervisor*, care așteaptă sosirea cererilor din partea clienților, și la fiecare cerere sosită, el va crea un nou proces fiu, un *worker* care va fi responsabil cu rezolvarea propriu-zisă a clientului respectiv, iar *supervisor*-ul va relua imediat așteptarea sosirii unei noi cereri, fără să aștepte terminarea procesului fiu. (Sau, alternativ, se poate implementa printr-un singur proces *multi-threaded*.)
- Pentru comunicarea între procesele client și procesul server este necesar să se utilizeze, drept canale de comunicație, fișiere *fifo*. (*Motivul*: nu se pot folosi canale anonime deoarece procesul server și procesele clienți nu sunt înrudite prin `fork/exec`.)
- Permisele fișierelor *fifo* folosite pentru comunicație trebuie configurate adecvat, astfel încât să permită execuția proceselor client din *conturi utilizator diferite* (!).





## Aplicația #2: implementarea unei aplicații client/server (cont.)

[Introducere](#)

[Canale anonime](#)

[Canale cu nume \(\*fifo\*\)](#)

[Caracteristici comune pentru ambele tipuri de canale](#)

[Șabloane de comunicație între procese](#)

[Aplicații ale canalelor de comunicație](#)

[Aplicația #1: implementarea unui semafor](#)

[Aplicația #2: implementarea unei aplicații client/server](#)

[Referințe bibliografice](#)

- Un alt aspect legat tot de comunicație: serverul nu cunoaște în avans clienții 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, și va aștepta sosirea informațiilor pe acest canal.

Un client oarecare se va conecta la acest canal *fifo* cunoscut și va transmite informații de identificare a sa, care vor fi folosite ulterior pentru realizarea efectivă a comunicațiilor implicate de serviciul solicitat (cel mai probabil va fi nevoie să utilizați canale suplimentare, particulare pentru acel client, pentru a nu se “amesteca” între ele comunicațiile destinate unui client cu cele destinate altui client conectat la server în același timp cu primul).

*Temă:* implementați un joc *multi-player* “în rețea”, pe baza ideilor descrise mai sus.





# Bibliografie obligatorie

Introducere

Canale anonime

Canale cu nume (fifo)

Caracteristici comune pentru ambele tipuri de canale

Șabloane de comunicație între procese

Aplicații ale canalelor de comunicație

Referințe bibliografice

[1] Cap. 5, §5.1, §5.2, §5.3 și §5.5 din cartea “*Sisteme de operare – manual pentru ID*”, autor C. Vidrașcu, editura UAIC, 2006. *Notă*: este accesibilă, în format PDF, din pagina disciplinei “Sisteme de operare”:

- <https://profs.info.uaic.ro/~vidrascu/S0/books/ManualID-S0.pdf>

[2] Programele demonstrative amintite pe parcursul acestei prezentări pot fi descărcate de la:

- <https://profs.info.uaic.ro/~vidrascu/S0/cursuri/C-programs/pipe/>
- <https://profs.info.uaic.ro/~vidrascu/S0/cursuri/C-programs/fifo/>

[3] Suportul online de laborator asociat acestei prezentări:

- [https://profs.info.uaic.ro/~vidrascu/S0/labs/suport\\_lab12.html](https://profs.info.uaic.ro/~vidrascu/S0/labs/suport_lab12.html)

## Bibliografie suplimentară:

[4] Cap. 44 din cartea “The Linux Programming Interface : A Linux and UNIX System Programming Handbook”, autor M. Kerrisk, editura No Starch Press, 2010.

- <https://profs.info.uaic.ro/~vidrascu/S0/books/TLPI1.pdf>

[5] POSIX API: `man 2 pipe`, `man 2 mkfifo`, `man 2 fcntl`.