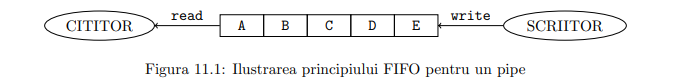
**Lab 11 – Comunicarea prin pipe-uri**

**11.2.1 Principii de comunicare**

🡪pipe = tip special de fisier utilizat pentru comunicarea intre procese

🡪 operatii : **read , write**

🡪 functioneaza dupa principiul **FIFO**

*~ scriere : din dreapta lui E*

*~ citire : din stanga lui A*

🡪conditii normale: la fiecare dintre capetele unui pipe este conectat un proces (unul citeste si altul scrie) 🡪 OBS: SO permite aceluiasi proces sa faca ambele operatii (rar in practica)

***🡪daca ambele capete sunt conectate:***

🡺 in momentul in care pipe-ul e gol **apelul de sistem READ NU va returna 0, ci se va bloca procesul apelant pana ce datele cerute sunt scrise**

🡺 in momentul in care pipe-ul e plin  **apelul de sistem WRITE va bloca procesul apelant pana ce in pipe va fi suficient spatiu liber pentru scrierea datelor**

**🡪 *daca se incearca citirea dintr-un pipe gol, dar nu avem nici un scriitor conectat,* cititorul NU va mai fi blocat, ci operatia READ va returna 0!!!**

🡪 ***daca se incearca scrierea intr-un pipe la care nu e conectat nici un cititor, citiroul NU va mai fi blocat, ci operatia de WRITE va primi semnalul SIGPIPE care in mod normal INCHIDE procesul respectiv(*** *obs: se poate preveni inchiderea receptionand explicit SIGPIPE)*

*🡪***la deschiderea unui pipe cu nume folosind open** **(citire sau scriere) PROCESUL CURENT E BLOCAT PANA CE ALT PROCES SE VA ATASA LA CELALALT CAPAT AL PIPE-ULUI , APELAND OPEN CU OPERATIA OPUSA!!!**

Text

Description automatically generated**11.2.2 pipe-uri cu nume**

**mkfifo 🡪 crearea unui pipe cu nume, primind calea si permisiunile de acces**

**🡪 in caz de succes returneaza 0**

**11.2.2.1 exemplu**

A picture containing text

Description automatically generated**OBS:** nu trebuie sa ne facem griji daca pipe-ul mai e utilizat de alt reader pentru ca la stergerea unui fisier deschis de anumite procese (desi numele lui dispare din arborele de fisiere) inode-ul si datele sunt pastrate pana ce toate procesele inchid fisierul.

🡪 singura problema ar fi daca pipe-ul ar fi sters de writer inainte de a fi deschis de reader 🡪 NU SE INTAMPLA ASTA

🡪 dupa compilarea celor 2 programe se executa prima data writer ce creeaza pipe-ul apoi **se blocheaza la apelul open** (lin 21) .

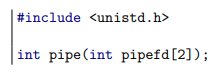
🡪 se continua prin executarea programului reader intr-un alt terminal. La apelul **open(lin 15) , pipe-ul va avea ambele capete deschise si ambele programe pot continua .**

**🡪**  din acest moment, **programele continua in ordine arbitrara**

**Obs:: daca reader ajunge primul la READ, acesta va fi blocat pana ce datele sunt scrise in pipe de catre writer**

**11.2.3 Pipe-uri anonime**

🡪 nu au nume dupa care sa se identifice, ele sunt accesibile doar prin descriproti de fisier **obtinuti prin apelul PIPE**



**🡪pipe:** 🡪 primeste ca argument un vector cu 2 elem de tip int in care vor fi completati cei 2 descr de fisier: **pipefd[0]🡪 CITIRE**  si **pipefd[1] 🡪 SCRIERE**

🡪se executa ulterior apelul **fork()** pentru a crea procesul copil ce va mosteni structurile de date ale parintelui **inclusiv descriptorii de fisiere** si astfel se va realiza comunicarea prin pipe-uri

**11.2.3.1 exemplu pipe-uri anonime**

Graphical user interface, text, application

Description automatically generatedText

Description automatically generated

🡪 procesul parinte initializeaza pipe-ul si sunt scrisi descriptorii in vectorul fd

🡪se creaza procesul copil ; ambele procese inchid capatul nefolosit al pipe-ului .

🡪 procesul parinte scrie ( close fd[0] ) si copilul citeste (close fd[1])

**11.2.4 Comunicarea bidirectionala**

🡪comunicare bidirectionala cu un singur pipe: **obligatoriu sincronizare explicita cu semafoare,** altfel se poate intampla ca un proces sa citeasca datele pe care tot el le-a scris

🡪 comunicare bidirectionala cu 2 pipe-uri

Graphical user interface, text, application

Description automatically generatedText

Description automatically generated

Table

Description automatically generated with medium confidence🡪**procesul parinte** initializeaza variabila cu cu ‘a’ si incepe o bucla de 10 iteratii in care:~transmite litera catre copil prin pipe-ul P2C

~citeste litera primita de la copil prinpipe-il C2P

~afiseaza litera si o incrementeaza

🡪 **procesul copil**  🡪 similar cu parintele dar incepe citirea literei prin pipe-ul P2C si se termina cu trimiterea literei incrementate catre parinte prin pripe-ul C2P

🡺 la executie: cele 2 procese ajung se afiseze alternativ primele 20 litere ‘a’->’t’