Sisteme de operare
Laborator 8
Comunicare inter-procese, pipe-uri

1. Creati un nou subdirector *lab8/* in structura de directoare a laboratorului creata anterior (*SO/laborator*) si subdirectoarele aferente *doc src* si *bin*. Nu uitati sa actualizati variabila de mediu PATH pentru a include directorul *SO/laborator/lab8/bin*.

2. Scrieti un program C **pipe.c** care creaza un *pipe* intre un proces parinte si un copil. Parintele scrie mesajul “hello world\n” in *pipe* si asteapta sa se termine procesul copil. Copilul citeste mesajul trimis de parinte in *pipe* si il afiseaza pe ecran. Primul lucru pe care il fac atat parintele cat si copilul dupa crearea *pipe*-ului este sa inchida partea de *pipe* pe care nu o folosesc la comunicarea dintre ei.

3. Scrieti un program C **pipe2.c** care modifica programul de la punctul 2 a.i. :
- procesul copil nu inchide capatul sau de scriere in *pipe*
- dupa ce a citit din *pipe* mesajul scris de procesul parinte, copilul scrie inapoi in *pipe* mesajul “done\n” si se termina
- dupa ce a scris in *pipe* mesajul “hello world\n” pentru copil, parintele asteapta sa citeasca din *pipe* mesajul “done\n” scris de copil si apoi il afiseaza pe ecran.

Ce se intampla? Cum explicati situatia?

4. Scrieti un program C **sigpipe.c** care creaza un *pipe* intre un proces parinte si un copil. Copilul inchide capatul sau de citire din *pipe*, simuleaza executia unui job folosind functia *sleep* pentru a dormi 1 secunda si se incheie. Parintele apeleaza si el *sleep* pentru 2 secunde dupa care scrie mesajul “hello world\n” in *pipe* si asteapta sa se termine procesul copil.

Ce se intampla? Cum explicati situatia? Dar daca programul prinde SIGPIPE ce se intampla?

5. Scrieti un program C **vfork-pipe.c** care creeaza un proces copil cu ajutorul apelului sistem *vfork* si creeaza un *pipe* intre cele doua procese. Procesul copil citeste un caracter din *pipe* si il afiseaza pe ecran impreuna cu PID-ul sau. Apoi incheie executia cu succes (cod zero). Procesul parinte scrie caracterul ‘b’ in *pipe* dupa care incheie executia.

Ce se intampla? Cum explicati situatia?

6. Scrieti un program C **ls-more.c** care emuleaza urmatoarea comanda shell:

$ ls -l | less

Programul creeaza in procesul parinte un *pipe* si doua procese copil, primul responsabil de executia comenzii *ls -l*, iar cel de-al doilea pentru comanda *less* (alegeti cea mai potrivita varianta de apel *exec* folosind exemplul comenzii emulate). Dupa ce a creat procesele copil, procesul parinte se blocheaza asteptand terminarea lor si recupereaza starea lor de terminare.

Indicatie: comanda *ls -l* tipareste output-ul pe ecran, in vreme ce *less* in comanda de mai sus citeste datele de intrare de la *stdin* (nu are ca parametru un nume de fisier). In aceasta situatie e nevoie sa redirectati din program *stdout*-ul lui *ls -l* catre capatul de scriere al *pipe*-ului, respectiv *stdin*-ul lui *less* la capatul de citire al *pipe*-ului. In acest sens, folositi *dup/dup2*.

Nu uitati sa inchideti in fiecare proces capetele de *pipe* care nu sunt folosite.

7. Scrieti un program C **my-mkfifo.c** care foloseste apelul *mkfifo* pentru a crea un *named pipe* (fisier de tip *FIFO*) cu numele dat ca argument in linie de comanda. Programul are efectul urmatoarelor comenzi shell:

```
$ mknod <nume fifo> p
```

sau

```
$ mkfifo <nume fifo>
```

Folositi comanda *ls -l* pentru a identifica daca fisierul creat are atribute de fisier *FIFO*.

ex:
```
$ mknod myfifo p
$ ls -l myfifo
```

Apoi, scrieti doua programe C **fifo-echod.c** si **fifo-echo.c** care scriu si citesc dintr-un fisier creat cu una dintre comenzile anterioare (fie cea rezultata prin compilarea programului **my-mkfifo.c**, fie prin comenzile shell de mai sus). **fifo-echod.c** deschide pentru citire si scriere fisierul *FIFO* primit ca parametru in linie de comanda si citeste in bucla caractere din fisierul *FIFO*. Dupa fiecare string de caractere citit din *FIFO*, **fifo-echod.c** trimite inapoi in *FIFO* string-ul citit (i.e., are functionalitate de serviciu de echo). Programul **fifo-echod.c** se lanseaza in background ca in exemplul de mai jos si asteapta sa primeasca mesaje in fisierul *FIFO*:

```
$ gcc -o fifo-echod fifo-echod.c
$ fifo-echod myfifo &
$ jobs
```

Programul **fifo-echo.c** deschide pentru citire si scriere fisierul *FIFO* primit ca parametru in linie de comanda si citeste in bucla caractere de la tastatura, de ex cu functia *fgets*. Dupa fiecare citire, programul scrie in *FIFO* caracterele citite de la tastatura, asteapta ecoul de la *echod* (i.e. citeste din *FIFO* ceea ce scrie *echod*) si tipareste caracterele pe ecran. Programul ruleaza la infinit pana cand e oprit cu Ctrl-C.

```
$ gcc -o fifo-echo fifo-echo.c
$ fifo-echo myfifo
```

Indicatie: pentru a scrie mai repede programele **fifo-echod.c** si **fifo-echo.c** refolositi codul de la programele **ping.c** si **pong.c** din laboratorul 4.