# Sisteme de operare
# Laborator 4
# File I/O (cont.)

**1.** Creati un nou subdirector *lab4/* 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/lab4/bin*.

**2.** Modificati programul **mycp.c** din laboratorul 2, care citeste in bucla date de la *stdin* (file descriptor 0) si le scrie la *stdout* (file descriptor 1) in sensul in care va primi doua argumente: numele fisierului sursa si numele fisierului destinatie. Fara sa modificati deloc bucla de copiere (adica file descriptorii folositi), deschideti corespunzator cele doua fisiere primite care argumente si folosind *dup2* faceti programul sa functioneze ca un program de copiere de fisiere.

*$ mycp fisier-sursa fisier-destinatie*

Ce se intampla daca fisierul sursa este chiar terminalul pe care lucrati, obtinut cu comanda *tty*? De exemplu:

*$ mycp `tty` fisier-destinatie*

**NB:** notatia unei comenzi intre backquotes ca mai sus instruieste shell-ul sa execute comanda mai intai si apoi sa inlocuiasca textul dintre backquotes cu rezultatul executiei comenzii. In particular, daca rezultatul comenzii *tty* de mai sus ar fi */dev/tty1*, shell-ul ar interpreta comanda de mai sus drept:

*$ mycp /dev/tty1 fisier-destinatie*

**3.** Scrieti un program C **temp.c** care simuleaza felul in care programele complexe gestioneaza fisierele temporare. In acest sens, mai intai creati un fisier *tempfile* cu urmatoarea comanda:

*$ echo "this is a temporary file" > tempfile*
*$ cat tempfile*
*$ ls -l tempfile*

Programul deschide fisierul *tempfile* folosind apelul sistem *open* apoi sterge fisierul *tempfile* folosind apelul sistem *unlink* si afiseaza pe ecran un mesaj din care sa reiasa ca fisierul *tempfile* a fost sters. Apoi simuleaza executarea unui task de 15 secunde apeland *sleep*. Dupa ce programul iese din *sleep*, citeste continutul fisierului *tempfile*, il afiseaza pe ecran si termina executia.

Pentru a intelege mai bine ce se intampla, rulati programul in background (folosind "&"). De exemplu, daca programul se numeste **temp.c** si va aflati in subdirectorul *src*:

*$ gcc -o ../bin/temp temp.c*
*$ temp &*

Dupa ce programul a afisat pe ecran notificarea stergerii fisierului *tempfile*, rulati comanda

*$ ls -l tempfile*

pentru a va convinge ca programul a fost intr-adevar sters. Asteptati 15 secunde. Ce se intampla? Cum va explicati efectul executiei programului daca reflectati la suita de operatii executate?

**4.** Scrieti un program C **myls.c** care afiseaza continutul unui director furnizat ca parametru al programului, simuland functionarea simpla, neparametrizata, a comenzii *ls*.

*Indicatie*: Folositi apelurile sistem *opendir*/*readdir*/*closedir*.

Folositi ulterior apelurile sistem *stat/fstat/lstat* pentru a implementa formatul lung al comenzii:

*$ myls -l <dirname>*

**5.** Scrieti un program C **mycd.c** care primeste un director ca parametru si schimba directorul curent la valoarea parametrului primit folosind *chdir*. Apoi, foloseste apelul sistem *getcwd* pentru a afla directorul curent si il tipareste pe ecran.

Testati programul utilizand urmatoarea secventa de comenzi:

*$ pwd*
*$ mycd /etc*
*$ pwd*

Ce observati? De ce e *cd* comanda interna a shell-ului?

**6.** Scrieti doua programe **ping.c** si **pong.c** care comunica printr-un fisier FIFO. Programele deschid fisierul FIFO atat pentru scriere cat si pentru citire. Programul **ping.c** scrie mesajul *“ping\n”* in fisier, doarme 5 secunde folosind apelul de biblioteca *sleep* (*man 3 sleep*), si apoi citeste din FIFO raspunsul programului **pong.c** pe care il afiseaza pe ecran. Programul **pong.c** citeste un mesaj din FIFO pe care il afiseaza pe ecran iar apoi raspunde cu mesajul *“pong\n”*.

*Indicatie*: Pentru a crea fisierul FIFO folositi comenzile *mknod/mkfifo*.