Utilizarea sistemelor de operare

nume: .....

e-mail: .....

data: ..... grupa: ..... nr. coli adaugate: .....

1. Ce se afisaza la rularea codului C de mai jos ?
Cate procese au aparut in total (incluzand si procesul initial) ?
Desenati schitat arborescenta acestor procese, indicand in dreptul fiecarui proces care a afisat ceva valoarea afisata (intre paranteze).
Justificati raspunsurile.

```
int x=0;
for(fork(); ! fork(); exit(x))
 for(! fork(); exit(x); fork())
  for(exit(x); fork(); ! fork());
printf("%d\n",++x);
```

2. Ce se afisaza la executarea urmatorului cod C, daca este lansat din linia de comanda shell cu './prog 5' (presupunand ca 'prog' este numele programului) ? Justificati raspunsul.

```
int main(int argc, char **argv) {
  int n;
  write(STDOUT_FILENO, "a", sizeof(char));
  n = atoi(argv[1]);
  if(n >= 3) { --n;  execv(argv[0], argv); }
  write(STDOUT_FILENO, "b", sizeof(char));
  return 0;
}
```

3. Ce se afisaza la rularea codului C de mai jos (presupunem ca timpii consumati de procese in afara lui 'sleep()' sunt neglijabili) ? Justificati raspunsul.

```
int d1, d2; char x, y;

d1 = open("f", O_CREAT | O_TRUNC | O_RDWR, S_IRWXU);
write(d1, "abcde", 5 * sizeof(char));
lseek(d1, 0, SEEK_SET);

if(fork()) {d2 = dup(d1); sleep(1);} else d2 = open("f", O_RDWR);

read(d1, &x, sizeof(char));  read(d2, &y, sizeof(char));

printf("%c\n%c\n", x, y);
```

4. Scrieti o functie in limbajul C:

```
long simetric(char const *fis)
```

care primeste ca argument specificatorul unui fisier si ii concateneaza la sfarsitul continutului o copie inversata a acestuia; de exemplu, daca fisierul continea 'abc', va contine in final 'abccba'.
Functia va opera direct pe fisierul respectiv, fara fisiere auxiliare iar memoria folosita va fi limitata de o constanta (in particular, nu se va incarca tot continutul fisierului in memorie). La sfarsit, functia va lasa fisierul inchis si va returna dimensiunea initiala a fisierului sau -1 in caz de eroare.

5. Scrieti un program C 'run' care se lanseaza in executie sub forma:

run director fisier

unde 'director' si 'fisier' sunt specificatorii (nume/cale) unui director, respectiv fisier obisnuit (regular), si care efectueaza urmatoarele:
- identifica fiecare fisier obisnuit (regular) care figureaza in 'director', care are ca proprietar proprietarul efectiv al procesului curent (adica 'run') si are setat cel putin un drept de executie;
- fiecarui fisier identificat mai sus, ii seteaza dreptul de executie pentru proprietar, apoi il lanseaza ca un proces copil avand iesirea standard redirectata catre 'fisier'; copii vor fi lansati a.i. sa se execute in paralel;
- dupa lansarea copiilor, 'run' asteapta terminarea lor si apoi se termina.

Putem presupune ca toti specificatorii de fisier sau director (cale/nume) au < 256 caractere. Programul va inchide explicit fisierele sau directoarele pe care le deschide.

Exemplu de testare:

```
$ls -l dir
total 48
-rw-rwxr-x 1 dragulici dragulici 16696 Jan  3 00:13 hello
-rw-rw-r-- 1 dragulici dragulici    62 Jan  3 00:12 hello.c
-rwxrwxr-x 1 dragulici dragulici 16696 Jan  3 00:14 salut
-rw-rw-r-- 1 dragulici dragulici    62 Jan  3 00:12 salut.c
$./run dir fis
$cat fis
Hello
Salut
$ls -l dir
total 48
-rwxrwxr-x 1 dragulici dragulici 16696 Jan  3 00:13 hello
-rw-rw-r-- 1 dragulici dragulici    62 Jan  3 00:12 hello.c
-rwxrwxr-x 1 dragulici dragulici 16696 Jan  3 00:14 salut
-rw-rw-r-- 1 dragulici dragulici    62 Jan  3 00:12 salut.c
$dir/hello
Hello
$dir/salut
Salut
```

6. Scrieti un program in limbajul C care efectueaza urmatoarele:
- genereaza doua procese copil conectate la parinte prin tuburi astfel:

copil1 --->--- parinte ---<--- copil2

fiecare copil scrie in tubul sau prin standard output iar parintele citeste din tuburi; copii isi vor inchide descriptorii nefolositi pe tuburi;
- dupa generarea copiilor, parintele citeste cu format intregi de la standard input, pana la terminarea intrarii, si scrie fiecare intreg in cele doua tuburi;
- copilul 1 citeste intregii din tubul sau si, pentru fiecare intreg citit, daca este impar, trimite parintelui un semnal SIGUSR1; la terminarea intrarii, se termina si el; copilul 2 procedeaza similar, dar testeaza daca intregii sunt pari si trimite parintelui semnale SIGUSR2;
- parintele, in timp ce citeste si scrie intregii, trateaza si semnalele primite si numara cate semnale a primit din fiecare tip; dupa ce a terminat de citit si scris intregii, inchide tuburile (ceea ce va semnala copiilor terminarea intrarii), apoi va astepta terminarea copiilor, va afisa numarul semnalelor primite din fiecare tip si se va termina.

Se va asigura protectia la pierderea semnalelor, prin blocarea/deblocarea semnalelor si asteptarea/trimiterea de semnale de raspuns (se va raspunde cu acelasi semnal care s-a primit); handlerele de semnal se vor instala a.i. primirea semnalului in parinte sa nu afecteze citirea/scrierea concomitente a intregilor (putem presupunem ca citirea/scrierea unui intreg se face atomic).

Exemplu testare (presupunem ca programul s.n.'prog'):

```
$cat fis.txt
12 42 13 4 55 22
$./prog < fis.txt
SIGUSR1 = 2, SIGUSR2 = 4
```