Enunt

Sa se implementeze un server de fisiere ce va folosi portul TCP 8192 pentru a raspunde la cereri din retea. Serverul trebuie sa ofere urmatoarele functii clientilor: listarea fisierelor dintr-un director, citirea dintr-un fisier, scrierea intr-un fisier.

Toate operatiile se vor face asincron folosind mecanismele cele mai scalabile oferite de fiecare sistem de operare.

Se va implementa de asemenea si un client, cu urmatoarea comportare:

- Sintaxa:

```
client adresa_server rd | wr fisier offset len
```

Actiune: citirea, respectiv scrierea in fisierul `fisier` `len` octeti incepand de la pozitia `offset`. Datele se vor citi de la STDIN si se vor scrie la STDOUT.

- si

```
client adresa_server ls director
```

Actiune: listarea fisierelor din directorul `director` la STDOUT. Pentru a usura testarea, am ales sa impunem un format simplu pentru listing, si anume un fisier/director pe fiecare linie precedat de marimea lui. Directoarele vor fi urmate de caracterul '/'. Marimea directoarelor este cea intoarsa de ls(0 pe Windows, un multiplu de block size pe Linux).

Precizari Generale

Ordinea parametrilor trebuie respectata, intrucat testele fac anumite presupuneri asupra lor.

Serverul trebuie sa asculte eventualele cereri pe adresa 127.0.0.1

Clientul va afisa la stdout numai ceea ce primeste de la server.

In cazul in care apar erori la server, acestea vor fi trimise clientului care va iesi cu un cod de iesire != 0.

De asemenea, daca clientul primeste parametri aiurea, trebuie sa iasa cu cod de iesire != 0.

FIXME: Pentru a evita un race in testare, este nevoie sa asteptati in client terminarea operatiei de scriere in server inainte de a iesi. Altfel, s-ar putea ca testul sa compare fisierul output cu cel corect inainte sa termine serverul de scris.

Exemple de functionare client:

```
[tema4]$ ./client 127.0.0.1 ls testdir 2>/dev/null
         0 dir1/
        24 testfile1
        12 x.txt
[tema4]$ ./client 127.0.0.1 rd testdir/testfile1 0 100 2>/dev/null
1234567890
abcdefghij
[tema4]$
[tema4]$ ./client 127.0.0.1 wr testdir/testfile1 4 6 2>/dev/null
abcdef
[tema4]$ cat testdir/testfile1
1234abcdef
abcdefghij
[tema4]$
```

Testare

Pentru simplificarea procesului de corectare al temelor, dar si pentru a reduce greselile temelor trimise, corectarea temelor se va face automat cu ajutorul unor teste publice (linux, windows).

Formatul afisarii, asa cum a fost descris mai sus, este obligatoriu.

In urma compilarii temei trebuie sa rezulte doua executabile: `server` (respectiv `server.exe` pentru Windows) si `client` (respectiv `client.exe` pentru Windows). Numele acestor executabile trebuie sa fie respectat.

Se vor acorda 0.4p pentru fiecare test. Nota rezultata poate fi modificata prin depunctari suplimentare:

- -0.1 pentru README necorespunzator
- -0.1 pentru surse prost comentate
- -0.2 pentru ilizibilitatea codului
- -0.2 neverificarea conditiilor de eroare/memory leak-uri/etc
- -0.4 implementare incorecta(protocol incorect, race-uri, potentiale erori de memorie)

Precizari Windows

Se vor folosi functii asincrone de citire/scriere pe socketi/fisiere impreuna cu mecanismul de completion ports pentru primirea notificarilor. Pool-ul de threaduri va fi egal cu numarul de procesoare din sistem. Numarul de procesoare se afla cu ajutorul functiei GetSystemInfo. Pentru listarea continutului unui director se pot folosi functiile FindFirstFile/FindNextFile.

Tema se va rezolva folosind doar functii Win32. Se pot folosi de asemenea si functiile de formatare `printf`, `scanf`, functiile de alocare de memorie `malloc`, `free` si functiile de manipulare a sirurilor de caractere (`strcat`, `strdup`, etc.)

Pentru partea de I/O si procese se vor folosi doar functii Win32. De exemplu, functiile `open`, `read`, `write`, `close` nu trebuie folosite, in locul acestor trebuind sa folositi `CreateFile`,`ReadFile`, `WriteFile`, `CloseHandle`.

Precizari Linux

Pe Linux, datorita faptului ca nu exista inca un mecanism asemanator completion ports ca pe Windows, se va folosi epoll pentru a multiplexa clientii intr-un singur thread si functiile posix asincrone pentru citire/scriere din fisiere. Notificarea terminarii operatiilor asincrone se va face folosind semnale. Nu uitati sa legati biblioteca realtime (-lrt) pentru a avea acces la functiile posix asincrone. Pentru listarea continutului unui director se pot folosi functiile din dirent.h, opendir/readdir. Pentru aflarea marimii unui director se poate folosi functia stat.

Tema se va rezolva folosind doar functii POSIX. Se pot folosi de asemenea si functiile de formatare `printf`, `scanf`, functiile de alocare de memorie `malloc`, `free` si functiile de manipulare a sirurilor de caractere (`strcat`, `strdup`, etc.)

Pentru partea de I/O si procese se vor folosi doar functii POSIX. De exemplu, functiile `fopen`, `fread`, `fwrite`, `fclose` nu trebuie folosite, in locul acestor trebuind sa folositi `open`,`read`, `write`, `close`.