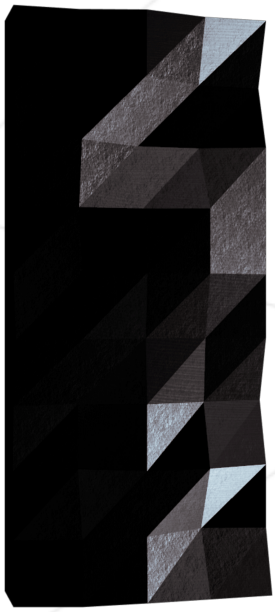

Proiect UNIX II

ft_p

42 staff staff@42.fr

Sumar: Acest proiect consta in implementarea unui client si a unui server ce permite transferul de fisiere in reteaua TCP/IP.

Cuprins

Capitolul I

Preambul

Iata ce spune Wikipedia in legatura cu subiectul Truffade:

Truffade este un fel de mancare pe baza de cartofi, branza de tip Salers, asezonata cu usturoi si sate si acompaniata de salata. Foarte des este servita jambon taranesc, acest fel se prepara uneri si cu cu branza tip tome proaspata sau de cantal. Truffade trebuie servit “dupa dorinta” in restaurante, intr-o caserola calda, pusa pe mijlocul mesei.

Etimologie

Truffade foloseste o ortografie anglicizata a cuvantului occitan “tRyfada” (trufada), in dialect. Radacina termenuluieste trufa sau trufla sau inca trûfét, semnificand cartofii, in dialectele auvergnat si rouergat.

Preparare

Cartofii sunt taiati in rondele si rumeniti intr-un vas in care in prealabil s-a topit untura alba. Se asezoneaza cu sare, piper si putin usturoi, desi, in Cantal, Rruffade se gateste fara usturoi. Cand cartofii sunt rumeniti, se stinge focul, se adauga branza taiata fasii. Se lasa branza sa se topeasca in contact cu cu cartofii fierbinti. Apoi se amesteca platoul timp de cinci minute. Apoi se pune totul intr-un vas incalzit in prealabil si se consuma cu muraturi locale.

Originile felului

Ciobanii catalani ce traiesc in mijlocul pasunilor se hranesc printre altele s cu cartofi, usor de conservat, o leguma raspandita pe tot teritoriul Frantei din secolul XIX. Probabil ca nu a trecut mult timp inainte ca unul dintre ei sa aiba ideea de a adauga branza pe care o produceau.

Capitolul II

Subiect - Parte obligatorie

Acest proiect consta in a creea un client si un server FTP (File Transfert Protocol) ce permite transmiterea si receptia fisierelor intre unul sau mai multi clienti si server.

Sunteti liberi liberi sa alegeti protocolul pe care sa-l utilizati (nu sunteti obligati sa respectati RFC ce defineste FTP, puteti inventa propriul protocol de transfer al fisierelor). Trebuie in schimb, indiferent de alegerea voastra, sa obtineti o coerenta intre client si server. Ei trebuie sa comunice corect impreuna.

Comunicatia intre client si server se face in TCP/IP (v4).

Capitolul III

Serveur

Utilizare:

```
$> ./server port
```

unde “port” corespunde numarului de port al serverului.

Serverul trebuie sa suporte mai multi clienti simultan prin intermediul unui fork.

Daca va simtiti confortabil cu select(2), il puteti folosi. Cu toate acestea aveti ocazia sa folositi irc-ul, profitati deci de proiect pentru a invata sa faceti un server care fork.

Daca folositi select(2), faceti lucrurile corect pana l acapat:

- select(2) la citire si scriere (nu intelegeti? folositi fork(2))
- server cenu se blocheaza (nu intelegeti? folositi fork(2)).

Daca serverul vostru este conceput gresit pentru ca ati vrut sa folositi select(2) in loc de fork(2), veti pierde multe puncte la sustinere.

Capitolul IV

Client

Utilizare:

```
$> ./client server port
```

oùunde “server” corespunde numelui gazda al masinii pe care se gaseste serverul si “port” este numarul portului.

Clientul trebuie sa cuprinda comenzile urmatoare:

- ls: lista directorului curent al serverului
- cd: schimba directorul curent al serverului
- get _file_: receptioneaza fisierul _file_ de la server la client
- put _file_: trimite fisierul _file_ de la client la server
- pwd: afiseaza calea spre directorul curent de pe server
- quit: taie legatura + iese din program

si sa raspunda la exigentele de mai jos:

- Un prompt specific spre client (pentru a distinge de Shell)
- E imposibil sa coborati la un nivel inferior, in ierarhia de directoare, si anume in directorul de executie al serverului (cel putin daca un parametru specificat pe server indica un alt director la pornire)
- Sa afiseze pe client mesajele SUCCES sau EROARE + explicatiile dupa fiecare cerere.

Capitolul V

Subiect - Parte bonus

Bonusul nu ava fi evaluat decat daca partea obligatorie este PERFECTA. Prin PERFECTA se intelege ca este realizata integral si ca functioneaza corect chiar daca se utilizeaza defectuos etc ... Concret, inseamna ca daca nu veti obtine TOATE punctele la partea obligatorie, partea bonus va fi IGNORATA.

Idei de bonus:

- lcd, lpwd si lls: aceste functii referitoare la sistemul de fisiere (filesystem) “local” si nu serve
- mget si mput: precum get si put dar “multiple”, pot contine “*”
- gestiunea login/password
- verificarea drepturilor
- posibilitatea specificarii unui director de baza diferit pentru fiecare login
- conversia ’\n’ in ’\r\n’ (unix<->windows) a fisierelor (modes “bin” si “asc”: binare = nu se converteste, ascii = conversie pe fisierul transferat)
- prompt
- respectarea RFC (standard 9 sau rfc 959)
- autocomplete la un get
- autocomplete la un put

Capitolul VI

Instructiuni

- Acest proiect nu va fi corectat de persoane umane. Sunteti deci liberi sa va organizati si sa denumiti fisierele asa cum doriti, respectand constrangerile de mai jos.
- Fisierul binar al aplicatiei server trebuie sa se numeasca `serveur`
- Fisierul binar al aplicatiei client trebuie sa se numeasca `client`
- Trebuie sa livrati un Makefile. El va trebui sa compileze clientul si serverul si sa contina regulile: client, server, all, clean, fclean, re. El nu trebuie sa recompileze fisierele binare decat in caz de necesitate.
- Proiectul trebuie sa fie scris conform standardului de cod (Norme).
- Trebuie sa tratati erorile intr-o maniera rezonabila. In niciun caz programul nu trebuie sa se termine intr-un mod neasteptat (Eroare de Segmentare etc...).
- Trebuie sa livrati in radacina directorului vostru de lucru/livrare un fisier `auteur` ce va contine login-ul vostru urmat de un '\n':

```
$>cat -e auteur
xlogin$
$>
```

- In cadrul partii obligatorii aveti dreptul sa folositi urmatoarele functii:
  - socket(2), open(2), close(2), setsockopt(2), getsockname(2)
  - getprotobyname(3), gethostbyname(3)
  - bind(2), connect(2), listen(2), accept(2)
  - htons(3), htonl(3), ntohs(3), ntohl(3)
  - inet_addr(3), inet_ntoa(3)
  - send(2), recv(2), execv(2), execl(2), dup2(2), wait4(2)
  - fork(2), getcwd(3), exit(3), printf(3), signal(3)
  - mmap(2), munmap(2), lseek(2), fstat(2)
  - opendir(3), readdir(3), closedir(3)
  - chdir(2), mkdir(2), unlink(2)
  - functiile autorizate in cadrul libft (read(2), write(2), malloc(3), free(3), etc... de exemplu ;-))
  - select(2), FD_CLR, FD_COPY, FD_ISSET, FD_SET, FD_ZERO insa doar daca sun folosite pentru a face lucrurile corect!

`ATTENTIE: asta nu are de a face cu socket-uri ne-blicante, care sunt interzise (deci fara fcntl(s, O_NONBLOCK))`

- Puteti sa folositi si alte functii in cadrul bonusului, cu conditia ca folosirea lor sa fie justificata corespunzator in timpul corectarii. Fiti isteti.
- Puteti pune intrebari pe forumul de pe intranet la sectiunea dedicata acestui proiect.

Capitolul VII

Notatare

- Notarea exercitiului `ft_p` se face in doi timpi:
  - Partea obligatorie: serveurul primeste 10 puncte si clientul tot 10
  - Partea bonus:
    * Nu ava fi evaluata deca daca partea obligatorie este PERFECTA (Totul trebuie sa functioneze corespunzator si tratarea erorilor sa fie fara greseala).
    * Veti obtine 1 si mai multe puncte pe functionalitati distincte de bonul si corect realizarte (Este la discretia corectorului vostru)
    * De asemenea, optimizarea calitatii anumitor elemente ale codului vor fi evaluate si pot primi puncte suplimentare.
- Succes tuturor!