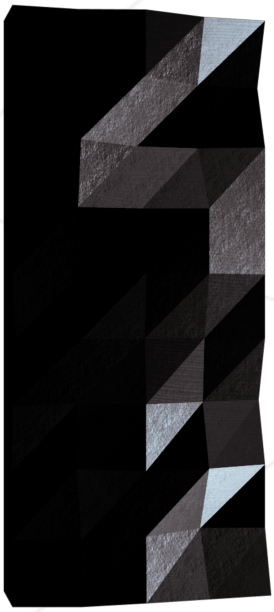

Projet UNIX et réseau

ft_p

42 staff staff@42.fr

Résumé: Ce projet consiste à implémenter un client et un serveur permettant le transfert de fichier en réseau TCP/IP

Table des matières

Chapitre I

Préambule

Voici ce que Wikipédia a à dire au sujet de la Truffade :

La truffade est un plat à base de pommes de terre, de fromage de type Salers, assaisonnée avec de l'ail et du sel et accompagnée de salade. Très souvent servie avec du jambon de pays, elle dérive parfois avec des fromage type tome fraîche ou de cantal (jeune). La truffade se doit normalement d'être servie "à volonté" dans les restaurants, dans une casserole encore chaude posée en milieu de table.

Étymologie

Truffade est une graphie francisée du vocable dialectal occitan "tRyfada" (trufada). La racine du terme trufada est trufa ou trufla ou encore trûfét , signifiant pomme de terre, patate, dans les dialectes auvergnat et rouergat.

Préparation

Les pommes de terre sont coupées en rondelles et rissolées dans une sauteuse dans laquelle on a préalablement fait fondre du lard blanc. On assaisonne avec du sel, du poivre et un peu d'ail, quoique, dans le Cantal, la truffade se cuisine sans ail. Lorsque les pommes de terres sont cuites, on coupe le feu, ajoute le fromage coupée en lamelles. On laisse le fromage fondre au contact des pommes de terre chaudes. Puis on remue le plat cinq minutes. Ensuite, on transvase le tout dans un plat préalablement chauffé et on déguste la préparation accompagnée de salaisons locales.

Origines du plat

Les bergers cantaliens vivant dans les burons au milieu des pâturages se nourrissaient entre autres de pommes de terre, faciles à conserver, un légume dont la culture se répand partout en France au cours du XIXe siècle. Il ne fallut probablement pas longtemps avant que l'un d'entre eux ait l'idée de les accompagner de la tome des fromages qu'ils fabriquaient.

Chapitre II

Sujet - Partie obligatoire

Ce projet consiste à faire un client et un serveur FTP (File Transfert Protocol) qui permettent d'envoyer et recevoir des fichiers entre un ou plusieurs clients et le serveur.

Vous êtes néammoins libre du choix du protocole à utiliser (vous n'êtes pas obligé de respecter les RFC définissant FTP, vous pouvez inventer votre propre protocole de transfert de fichier). Vous devrez par contre quelque soit votre choix absolument obtenir une cohérence entre votre client et votre serveur. Ils doivent communiquer correctement ensemble.

La communication entre le client et le serveur se fera en TCP/IP (v4).

Chapitre III

Serveur

Usage :

```
\$> ./serveur port
```

où “port” correspond au numéro de port du serveur.

Le serveur doit supporter plusieurs clients simultanément par l'intermédiaire d'un fork.

Si vous êtes très à l'aise avec select(2), vous pouvez l'utiliser. Néammoins vous aurez l'occasion de l'utiliser pour le projet irc, donc profitez de ce projet pour apprendre à faire un serveur qui fork.

Si vous utilisez select(2), faites les choses correctement jusqu'au bout :

- select(2) en lecture + écriture (vous ne comprenez pas ? utilisez fork(2))
- serveur absolument non bloquant (vous ne comprenez pas ? utilisez fork(2)).

Si votre serveur est mal conçu parce que vous avez voulu utiliser select(2) au lieu de fork(2), vous perdrez beaucoup de points en soutenance.

Chapitre IV

Client

Usage :

```
\$> ./client machine port
```

où “machine” correspond au nom d’hôte de la machine sur laquelle se trouve le serveur et “port” le numéro de port.

Le client doit comprendre les commandes suivantes :

- ls : liste le répertoire courant du serveur
- cd : change le répertoire courant du serveur
- get _file_ : récupère le fichier _file_ du serveur vers le client
- put _file_ : envoi le fichier _file_ du client vers le serveur
- pwd : affiche le chemin du répertoire courant sur le serveur
- quit : coupe la connection + sort du programme

et répondre aux exigences ci-dessous :

- Un prompt spécifique au client (pour le distinguer du Shell)
- Impossibilité de descendre à un niveau inférieur au répertoire d’exécution du serveur (à moins qu’un paramètre spécifié au serveur indique un autre répertoire de départ)
- affichage sur le client des messages SUCCESS ou ERROR + explication après chaque requête.

Chapitre V

Sujet - Partie bonus

Les bonus ne seront comptabilisés que si votre partie obligatoire est PARFAITE. Par PARFAITE, on entend bien évidemment qu'elle est entièrement réalisée, et qu'il n'est pas possible de mettre son comportement en défaut, même en cas d'erreur aussi vicieuse soit-elle, de mauvaise utilisation, etc ... Concrètement, cela signifie que si votre partie obligatoire n'est pas validée, vos bonus seront intégralement IGNORÉS.

Des idées de bonus :

- lcd, lpwd et lls : ces fonctions concernent le filesystem "local" et non serveur
- mget et mput : comme get et put mais "multiple", peut contenir des "*"
- gestion du login/password
- vérification des droits
- possibilité de spécifier un répertoire de base différent pour chaque login
- conversion des '\n' en '\r\n' (unix<->windows) des fichiers (modes "bin" et "asc" : binaire = pas de conversion, ascii = conversion sur le fichier transféré)
- prompt
- respect de la RFC (standard 9 ou rfc 959)
- support de l'IPv6
- complétion lors d'un get
- complétion lors d'un put

Chapitre VI

Consignes

- Ce projet ne sera corrigé que par des humains. Vous êtes donc libres d'organiser et nommer vos fichiers comme vous le désirez, en respectant néanmoins les contraintes listées ici.
- Le binaire du serveur doit se nommer `serveur`
- Le binaire du client doit se nommer `client`
- Vous devez coder en C et rendre un Makefile. Il devra compiler le client et le serveur, et contenir les règles : client, serveur, all, clean, fclean, re. Il ne doit recompiler les binaires qu'en cas de nécessité.
- Votre projet doit être à la Norme.
- Vous devez gérer les erreurs de façon raisonnée. En aucun cas votre programme ne doit quitter de façon inattendue (Segmentation fault, etc...).
- Vous devez rendre, à la racine de votre dépôt de rendu, un fichier `auteur` contenant votre login suivi d'un '\n' :

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

- Dans le cadre de votre partie obligatoire, vous avez le droit d'utiliser les fonctions suivantes :
  - socket(2), open(2), close(2), setsockopt(2), getsockname(2)
  - getprotobyname(3), gethostbyname(3), getaddrinfo(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)
  - les fonctions autorisées dans le cadre de votre libft (read(2), write(2), malloc(3), free(3), etc... par exemple ;-))
  - select(2), FD_CLR, FD_COPY, FD_ISSET, FD_SET, FD_ZERO mais uniquement si c'est pour faire quelque chose de correct !

`ATTENTION: cela n'a rien à voir avec des sockets non bloquantes qui elles sont interdites (donc pas de fcntl(s, O_NONBLOCK))`

- Vous avez l'autorisation d'utiliser d'autres fonctions dans le cadre de vos bonus, à condition que leur utilisation soit dûment justifiée lors de votre correction. Soyez malins.
- Vous pouvez poser vos questions sur le forum dans l'intranet dans la section dédiée à ce projet.

Bon courage à tous !