



LO14 : Administration des systèmes

Rapport du projet

Création d'un serveur d'archives

Automne 2021

Réalisé par Alexis Ta et Corentin Réault

SOMMAIRE

Introduction	3
Fonctionnement	3
La fonction interaction()	3
Le client vsh.sh()	5
list()	7
create ()	8
browse ()	12
a) touch()	13
b) cat()	15
c) pwd	18
c) cd	18
d) ls	20
extract()	22
Analyse	26
Conclusion	26

Introduction

Le projet LO14 a pour objectif de créer un serveur d'archive. Il s'agit de créer une nouvelle commande utilisateur permettant de communiquer avec un serveur. Ce serveur doit être disponible en permanence et doit pouvoir communiquer au travers de différentes commandes entrées par l'utilisateur. Le script utilisateur et serveur utilisent l'outil Netcat. Ils communiquent avec un tube qui sera défini par le serveur. Le tube sera détruit lorsque le script côté client sera interrompu ou terminé. De plus, nous voulons que plusieurs utilisateurs puissent se connecter simultanément au serveur. Il faudra donc attribuer un tube pour chaque client se connectant au serveur. Le serveur fonctionne en mode listening. Il attend les requêtes client. Afin de travailler de manière organisée, un GitHub est créé pour synchroniser et éditer différentes versions du projet. Discord et Facebook Messenger sont également employés pour communiquer et échanger des idées. Enfin, le code de base du serveur fourni en TD est utilisé pour des raisons pratiques. Ce code permet d'obtenir un serveur bash et d'établir une connexion TCP client/serveur. Il s'agit ainsi de la base sur laquelle repose notre projet. Le projet est disponible via ce lien : <https://github.com/UnderBinaire/lo14>.

Fonctionnement

Les commandes tapées doivent respecter une certaine syntaxe et avoir les caractéristiques suivantes. Le serveur est interrogé par l'intermédiaire d'une commande shell nommée vsh, qui fonctionne selon les modes suivants : Le mode list, le mode create, le mode browse et enfin le mode extract. Ces fonctions seront expliquées plus loin dans le rapport. Un script vsh est lancé pour permettre au client d'interagir avec le serveur. Au travers de ce rapport, nous détaillerons chacune de ces fonctions et nous expliquerons leur fonctionnement avec des captures d'écrans du code.

La fonction interaction()

Avant d'aller plus loin, il est nécessaire de permettre l'interaction entre le serveur et le client. Il faut que le serveur puisse reconnaître les commandes tapées par le client et il faut que le client puisse recevoir les réponses du serveur. Pour ce faire, une boucle while est créée. Cette boucle while tourne de manière indéfinie de sorte à garder le serveur à l'écoute en permanence. On initialise les variables browseMode à FALSE et currentDirectory à 0 (pour la détection d'erreur). Si l'utilisateur est en mode browse, dans ce cas, browseMode=TRUE et le serveur affiche « vsh :> » indiquant qu'il est en attente de l'utilisateur. La fonction interaction() permet aussi de distinguer les fonctions qui peuvent être lancées en mode Browse et le reste qui peuvent et doivent être lancées en dehors du mode browse. Par ailleurs, il est possible de quitter le mode browse à tout moment en tapant exit sur le terminal client. Enfin, la fonction garantit que l'utilisateur tape les bonnes commandes en renvoyant un message de confirmation.

Par ailleurs, lors de l'utilisation de la commande vsh, nous avons essayé d'utiliser une commande comme suit : "cat file | nc 'host' 'port'" ou "echo "browse archive" | nc 'host' 'port"'; or l'entrée standard devient donc le contenu renvoyé par le pipe, ce qui rend impossible les interactions à l'aide du terminal. Pour contourner ce problème, nous avons décidé d'utiliser un canal dédié à la transmission de la première commande du client vsh en lançant un nouveau netcat en mode listening sur le port 8082. Après la première commande, l'utilisateur peut interagir via le terminal (utile pour le mode browse).

```

● ● ●

function interaction() {
    local cmd args
    x=0
    while true; do
        if [[ $x -eq 0 ]]; then
            cmd=$(nc -l -p 8082)
            if [[ ! -z $cmd ]]; then
                args=$(echo $cmd | cut -d" " -f2)
                cmd=$(echo $cmd | cut -d" " -f1)
            fi
        else
            read -r cmd args || exit -1
        fi
        if $browseMode; then #Si browsemode est vrai (Si l'user invoque la fonction browse)
            fun="browse-$cmd"
        elif [[ -z $cmd && $x -eq 0 ]]; then
            fun="commande-browse"
            echo "$x $fun"
            echo "args recuper : $args"
        else
            fun="commande-$cmd"
        fi
        if [ "$(type -t $fun)" = "function" ]; then
            $fun $args
        elif [ "$fun" = "browse-exit" ]; then #Si je quitte browsemode
            browseMode=false
        else
            commande-non-comprise $fun $args
        fi
        if $browseMode; then
            echo -n "vsh:>"
        fi
        ((x++))
    done
}

```

Le client vsh.sh()

Le client vsh sert uniquement au client, et lui permet de transmettre des ordres au serveurs, d'envoyer et de recevoir des archives. Ce client détermine les fonctions choisies par l'utilisateur en fonction du nombre d'arguments fournis et par comparaison entre la chaîne de caractère issue du premier argument et des commandes spécifiques nécessitant un traitement particulier. Le fonctionnement de ce script repose sur l'utilisation de childs process, permettant l'exécution simultanée de plusieurs commande (via l'utilisation de '&')

```

#!/bin/bash

# Ce script est la partie client : lancé par la commande "vsh"

function sendCommand() {
    printf "%s\n" "$2 $3" | nc -c $1 8082
}

function transfertFile() {
    sleep 2
    dir=$(pwd | rev | cut -d'/' -f1 | rev)
    cd ..
    tar Jcvf - $dir | nc -c $1 8081 1> /dev/null
    rm receive.tar.xz
    cd $dir
}

function receiveFile() {
    sleep 2
    timeout 3s nc -c $1 8081 > send.tar.xz
}

commande=$(echo $1 | cut -c 2-)
if [ $# -eq 4 ]; then
    if [[ $commande = "create" ]]; then
        sendCommand $2 $commande $4 & transfertFile $2 & timeout 6s nc $2 $3
    elif [[ $commande = "extract" ]]; then
        sendCommand $2 $commande $4 & receiveFile $2 & timeout 6s nc $2 $3
        tar Jxvf send.tar.xz
        rm send.tar.xz
    elif [[ $commande = "browse" ]]; then
        sendCommand $2 $commande $4 & nc $2 $3
    else
        sendCommand $2 $commande $4 & nc $2 $3
    fi
elif [ $# -eq 3 ]; then
    sendCommand $2 $commande & timeout 0.5s nc $2 $3
else
    echo "commande interrompue, nombre d'arguments invalide"
fi

```

1. list()

La première fonction est la fonction list(). Elle a pour objectif d'afficher la liste des archives présente sur le serveur sur la machine cliente. L'utilisateur appelle la fonction en tapant simplement la commande list au clavier. Ensuite, le serveur effectue la commande « ls » pour capturer la liste des archives présentes dans un dossier archive créé manuellement par l'administrateur du serveur puis stocke cette liste dans une variable. Si cette variable correspond à une chaîne vide, alors il n'y a aucune archive. Sinon, les archives sont affichées sur le terminal client.

```
function commande-list() {  
    ls=$(ls archives)  
    if [ -n "$ls" ]; then  
        echo "Les archives présentes sur le serveur sont :"  
        echo $ls  
    else  
        echo "Pas d'archives présente sur le serveur"  
    fi  
}
```

2. create ()

La fonction create() crée une archive dont le nom est donné en argument. Il s'agit d'une archive contenant toute l'arborescence des répertoires et des fichiers contenus dans le répertoire courant de la machine cliente.

Tout d'abord la fonction vérifie que l'archive à créer n'existe pas déjà sur le serveur via l'utilisation de la commande 'ls' et d'une boucle for, permettant de comparer tous les résultats retournés par la commande 'ls' au nom de l'archive demandé par l'utilisateur.

Si une archive contenant ce nom existe sur le serveur, alors on retourne une erreur.

Sinon on transfère les fichiers du client au serveur en lançant netcat en mode listening côté serveur sur le port 8081 et en attendant la connexion du client vsh.

Une fois l'archive compressée reçue, on la décomprime dans le répertoire utilisateur et on supprime cette archive compressée, et on vérifie que le répertoire n'est pas vide.

Si celui-ci est vide on affiche une erreur sinon on crée les premières lignes de l'archive grâce à un simple 'ls' et une commande awk.

```

● ● ●

function commande-create() {
    echo "Mode create"
    nomArchive=$1 #Argument (Nom de l'archive)
    if [ -n "$nomArchive" ]; then
        occurrence=false #Flag false
        for i in $(ls archives); do
            if [ "$nomArchive" = "$i" ]; then
                occurrence=true
                break
            fi
        done

        if $occurrence; then
            echo "création impossible, une archive porte déjà ce nom"
        else
            echo "reception des fichiers"
            cd tmp_receive
            nc -l -p 8081 > receive.tar.xz
            tar Jxvf receive.tar.xz
            rm receive.tar.xz
            cd ..
        fi
    else
        echo "arborescence bien reçue"
        echo "3:5" >> archives/$nomArchive #Sinon, on met tout en haut du fichier 3:5 comme
dans l'énoncé et on crée automatiquement le fichier avec >>
        dir=$(ls tmp_receive)
        ls -l tmp_receive/$dir | awk -v dir=$dir 'BEGIN{print "\ndirectory
$dir}NR>1{n=split($1,tab,""); if(tab[1]==")print $9" "$1" "$5;"' >> archives/$nomArchive
        ls -l tmp_receive/$dir | awk 'NR>1{n=split($1,tab,""); if(tab[1]=="-")print $9"
"$1" "$5;}'END{print "@"}' >> archives/$nomArchive
        export -f addDir
        awk -v nomArchive=$nomArchive '{if($1==")directory"){dirArborescence=$2; gsub(/\\"/, "
/", dirArborescence); getline; while($1!="@"){n2=split($2,tab2,""); if(tab2[1]==")"
{cmd="addDir \"dirArborescence\" \"$1\" \"nomArchive\" \"dirArborescence\" \"$1\""; system(cmd);};"'
        getline}}' archives/$nomArchive
    ...
}
  
```

Pour la suite, on automatise la création de l'archive en recherchant les lignes dont le second champ commence par un ‘d’ entre chaque “directory” et le ‘@’ suivant, ce qui indique la présence d'un dossier, que l'on créera à la suite de l'archive en faisant appel à une fonction, qui permettra aussi d'ajouter le contenu de ce nouveau dossier.

Ainsi, comme nous ajoutons des lignes au fichier lu par awk, celui-ci pourra vérifier en passant aux lignes suivantes, si il y de nouveaux dossiers à créer parmi les lignes ajoutées.



```
function addDir() {
    dirArborescence=$(echo $1 | sed 's/\//\\/g')
    newDirName=$2
    nomArchive=$3
    cheminFichier=tmp_receive/$4
    echo "directory $dirArborescence\\$newDirName" >> archives/$nomArchive
    ls -l $cheminFichier | awk 'NR>1{n=split($1,tab,""); if(tab[1]==d)print $9" "$1" "$5;}' >> archives/$nomArchive
    ls -l $cheminFichier | awk 'NR>1{n=split($1,tab,""); if(tab[1]==-)print $9" "$1" "$5;}END{print "@"}' >> archives/$nomArchive
}
```

Après avoir créé le header de l'archive, on détermine la racine de l'archive, qui est le premier répertoire de l'arborescence à contenir des fichiers, et on lui ajoute un '\ à la fin de son chemin dans l'archive.

Nous modifions aussi la première ligne de l'archive, permettant de déterminer le début et la fin du header.

Ensuite on ajoute les données de chaque fichier à la fin de l'archive en modifiant dans le header la ligne correspondant au fichier pour ajouter les deux nombres de la partie data.

```

...
root=$(cat archives/$nomArchive | awk 'BEGIN{ligneCandidate=""; nbrChamps=""}
m=0}NR>2{if($1=="directory"){line=$0; n=split($2,tab,"\\");getline; if($1!="@"
{ligneCandidate=ligneCandidate" "line; nbrChamps=nbrChamps" "n;
m=m+1}}END{split(ligneCandidate,tab2," "); split(nbrChamps,tab3," "); min=100; indice=1;
for(i=1; i<=m; i++){if(tab3[i]<min){min=tab3[i]; indice=i}}indice=indice*2; print
tab2[indice]}'')
sed -i "s/^directory $root$/&\\\/g" archives/$nomArchive

nbrLinesHeader=$((wc -l archives/$nomArchive | cut -d' ' -f1)
((nbrLinesHeader++))
sed -i "ls/3:5/3:$nbrLinesHeader/" archives/$nomArchive

echo "traitment des données de chaque fichier"
while read -r line; do
    if [[ -z $(echo $line | grep '^@') ]]; then
        if [[ -z $(echo $line | grep '^directory') ]]; then
            type=$(echo $line | awk '{split($2,tab,""); print tab[1]}')
            if [ ! -z $(echo $type | grep '^-' ) ]; then
                fileName=$(echo $line | awk '{print $1}')
                startLine=$((wc -l archives/$nomArchive | cut -d' ' -f1)
                ((startLine++))
                nbrLinesFile=$((wc -l $chemin/$fileName | cut -d' ' -f1)
                ((endLine=nbrLinesFile-1))
                sed -i "s/^$fileName.*$/& $startLine $endLine/g" archives/$nomArchive
                cat $chemin/$fileName >> archives/$nomArchive
            fi
        else
            chemin=$(printf "%s\n" "$line" | awk '{gsub(/\//, "/", $2); print $2}')
            #chemin=$(printf "%s\n" "$chemin" | sed 's/\//\//g')
            chemin=tmp_receive/$chemin
        fi
    fi
done < archives/$nomArchive

echo "Le fichier a été créé avec succès"
fi
fi
rm -r tmp_receive/*
else
    echo "création impossible, pas de nom fourni pour l'archive"
fi
}

```

3. browse ()

La fonction browse() permet à l'utilisateur de naviguer dans une archive et de réaliser des opérations sur cette archive. On crée une variable browseArchive ayant pour valeur l'argument donné par l'utilisateur (le nom de l'archive dans laquelle l'utilisateur souhaite effectuer des manipulations). Un test est effectué au préalable afin de vérifier que l'utilisateur saisit une archive existante dans le serveur. Pour trouver une occurrence de cette archive, la fonction grep avec l'option -c est utilisée. Si le nombre d'occurrences est différent de 0, dans ce cas, le mode browse est activé et la variable globale browseMode prend la valeur TRUE. Ceci permet de rentrer en mode browse pour pouvoir effectuer des commandes relatives au mode browse tel que touch par exemple. Enfin, avec la fonction egrep et l'utilisation d'une expression régulière, on récupère la racine de l'archive dont il est question avec la fonction cut. Ceci permet aussi de créer le répertoire courant du mode browse qui est indispensable lors de l'utilisation de fonctions pour naviguer dans une archive. Ces fonctions associées au mode browse sont détaillées ci-dessous.

```
function commande-browse() {
    browseArchive=$1 #Argument
    if [ -n "$browseArchive" ]
    then #Si argument est une chaîne non vide (Donc il existe)
        trouve=$(ls archives | grep -c $browseArchive)
        if [ $trouve -eq 0 ]
        then
            echo "Navigation impossible, aucune archive de ce nom sur le serveur"
        else
            browseMode=true
            browseRoot=$(egrep '^directory.*\\$' archives/$browseArchive | cut -d" " -f2)
            currentDir=$browseRoot
        fi
    else
        echo "Navigation impossible, pas de nom fourni pour l'archive"
    fi
}
```

a) touch()

La fonction touch permet de créer une entrée dans l'archive pour un fichier vide dont le nom est renseigné en argument de la fonction si le fichier n'existe pas déjà. Pour commencer, on stocke le chemin du fichier entré par l'utilisateur dans l'archive. Cela donne la ligne où insérer le fichier vide dans le texte. Si l'argument est vide, dans ce cas il y a erreur. Sinon, on récupère le nom du fichier en utilisant la fonction rev pour inverser le texte du chemin du fichier pour pouvoir récupérer avec cut le nom du fichier entré par l'utilisateur (dernier champ de l'argument) et on inverse une nouvelle fois pour obtenir le chemin originel. Nous vérifions que le fichier que l'utilisateur souhaite créer n'existe pas déjà au préalable dans l'archive avec la fonction grep -c qui va compter le nombre d'occurrence du nom du fichier. Si il existe, la fonction s'arrête. Sinon, on récupère l'arborescence du chemin de façon analogue au nom du fichier puis nous faisons en sorte d'éviter les erreurs d'interprétation du script shell par rapport à l'utilisation du caractère '\` dans l'archive. On ajoute des caractères '\` pour échapper le caractère spécial avec sed substitute. Ensuite, l'existence de l'arborescence est vérifiée car il faut qu'elle existe avant de pouvoir insérer un fichier. En outre, la fonction stocke dans la variable match le dernier champ de l'arborescence afin de permettre de localiser la ligne sous laquelle le fichier doit s'insérer. Après cela, il suffit d'insérer le fichier vide au bon endroit avec sed -i /\(\$match\)\\$/a. On insère une ligne composée du nom du fichier et des droits correspondants à un fichier vide par défaut suivi de 0 pour la taille et pour les informations complémentaires puisqu'il s'agit d'un fichier sans contenu. Le chemin de l'archive est donné par la variable path. Cette variable permet ainsi d'éditer le fichier texte.

```

● ● ●

function browse-touch() {
    path="archives/"$browseArchive
    cheminFichier=$1
    if [ -z $cheminFichier ]
    then
        echo "Erreur, argument manquant"
    elif [ -n $cheminFichier ]
    then
        nomFichier=$(echo $cheminFichier | rev | cut -d"\\" -f 1 | rev) # Récupération du nom du
        fichier entré par l'user ( Dernier champ )
        occurrence=$(grep -c $nomFichier $path) #Chercher si le fichier existe dans le fichier texte (
        Je cherche si le nom apparaît dans une ligne )
        if [[ $occurrence -ne 0 ]]
        then
            echo "Le fichier existe déjà dans l'archive"
            return
        else
            arbo=$(echo $cheminFichier | rev | cut -d "\\" -f2- | rev) #Récupération de l'arbo donné
            par l'user avec des commandes CUT
            replace=$(echo $arbo | sed 's/\\\\\\\\\\//g') #Echapper les '\' pour éviter les bugs avec Grep
            occurrence2=$(grep -c $replace $path) #Je regarde s'il y a occurrence de l'arbo dans le
            fichier texte
            match=$(echo $replace | rev | cut -d"\\" -f1 | rev) #Prendre le dernier champ de l'arbo
            pour que sed puisse match une ligne ET une seule
            if [[ $occurrence2 -ne 0 ]] #Si occurrence est différent de 0, l'arbo existe (Chaîne
            trouvée) donc je peux insérer mon fichier
            then
                sed -i "/\($match\)\\\\*$a $nomFichier -rw-rw-r-- 0 0 0" "$path" #Insertion du
                fichier vide au bon endroit (En dessous de la ligne finissant par $match regex)
                echo "Fichier vide inséré dans l'archive avec succès !"
                header=$(head -1 $path | cut -d ":" -f2)
                let header2="$header"+1
                sed -i "1s/$header/$header2/" $path #Augmenter compteur header car insertion fichier
            else
                echo "L'arborescence décrite n'existe pas dans l'archive"
                return #Pas d'arbo existente donc erreur
            fi
        fi
    fi
}

```

b) cat()

La fonction cat() affiche le contenu de deux fichiers maximum. Le chemin est fourni en argument. Si l'utilisateur entre 1 fichier, le script récupère le nombre de lignes du header et stocke cette valeur dans la variable header. Le script récupère aussi le nombre de lignes total de l'archive avec la commande wc -l appliquée au fichier donnée par la variable "path". Le nombre de lignes du body est obtenu par simple différence entre le nombre total de lignes et le nombre de lignes du header. Le contenu du body est récupéré avec la commande tail en partant de la fin de l'archive afin d'obtenir toutes les lignes appartenant au body. Ensuite, de manière analogue aux autres fonctions, on récupère le nom du fichier, on vérifie que ce fichier existe, on récupère la taille du fichier, le type et on effectue des vérifications. S'il s'agit d'un dossier, il y a erreur. Si c'est un fichier vide, aucun contenu ne peut être affiché. Si il n'y a aucune occurrence du fichier, le fichier n'existe pas. Si tout est vérifié, on récupère le contenu des fichiers de manière analogue à la fonction extract() puis on les affiche sur le terminal client. Pour afficher 2 fichiers, on effectue les mêmes manipulations.

```

function browse-cat() {
    path="archives/$browseArchive
    if [ $# -eq 1 ]
    then
        header=$(head -1 $path | cut -d":" -f2) #Nombre de lignes du header
        w=$(cat "$path" | wc -l) #Nbre de lignes total de l'archive
        let body="$w-$header"+2 #Nombre de lignes du body
        contenu=$(cat "$path" | tail -$body) #Récupération du body
        fichier1=$1
        nomfich=$(echo "$fichier1" | rev | cut -d"\\" -f1 | rev) #Récuperer nom fichier entré par
        l'utilisateur
        occurrence=$(grep -c $nomfich $path) # Chercher si le fichier existe dans l'archive
        taille=$(cat "$path" | grep "^$nomfich" | cut -d" " -f3) #Récuperer la taille du fichier
        type=$(cat "$path" | grep "^$nomfich" | cut -d" " -f2) #Récuperer le type (si fich ou doss)
        if [[ "$taille" -eq 0 ]] && [[ $occurrence -ne 0 ]]
        then
            echo "Fichier vide. Aucun contenu à afficher"
            return
        fi
        if [[ "$type" == "d"* ]]
        then
            echo "Erreur, il s'agit d'un dossier"
            return
        fi
        if [[ $occurrence -eq 0 ]]
        then
            echo "Fichier n'existe pas"
            return
        else
            arbo=$(echo "$fichier1" | rev | cut -d"\\" -f2- | rev) #Verifier que arbo existe
            replace=$(echo "$arbo" | sed 's/\\\\\\\\\\g') #Echapper les \
            occurrence2=$(grep -c $replace $path) #Occurence de l'arbo ?
            if [[ $occurrence2 -eq 0 ]]
            then
                echo "Arbo existe pas"
                return
            else
                bodycommence=$(cat "$path" | grep "^$nomfich" | cut -d" " -f4)
                let bodycommence="$bodycommence"+1
                bodyetendre=$(cat "$path" | grep "^$nomfich" | awk '{print $5}') #Recuperer pour la
                ligne le nombre de lignes du body
                let bodyetendre="$bodyetendre"+1
                afficher=$(echo "$contenu" | sed -n "$bodycommence,$bodyetendre p") # J'affiche le
                contenu du fichier
                echo "Voici l'archive souhaitée :"
                echo "$afficher"
            fi
        fi
    fi
}

```

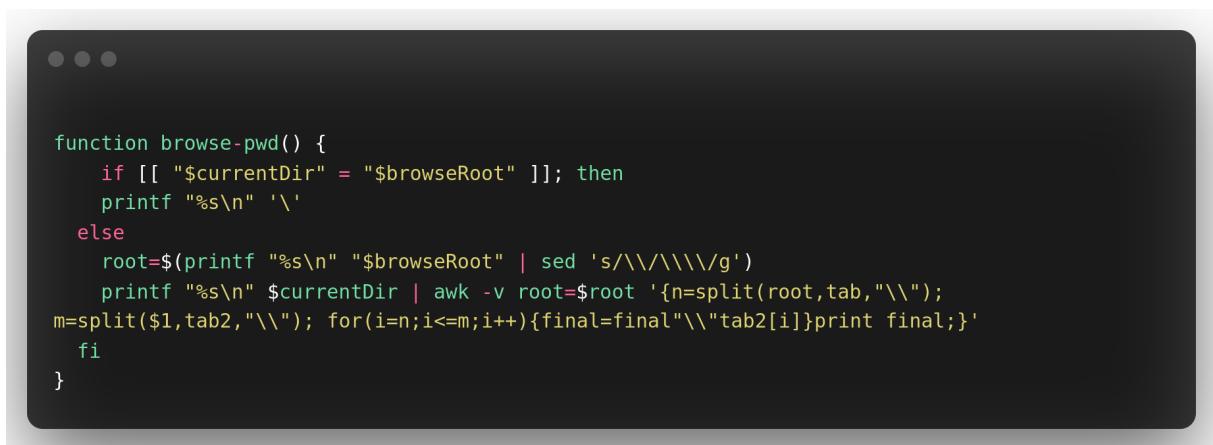
```

... (code continues)

```

c) pwd

Affiche le répertoire courant de l'utilisateur dans l'archive. Le début de l'arborescence est défini par la variable globale \$browseRoot qui contient le chemin du dossier racine de l'archive. Lorsque l'utilisateur arrive en mode browse, le répertoire courant défini par défaut est la racine de l'archive, affichée "/" si l'utilisateur effectue la commande pwd. Sinon l'on s'est déplacé dans l'archive, pwd affiche le chemin du répertoire courant à partir de la racine (ex : "/exemple1/exemple2")



```
function browsePwd() {
    if [ "$currentDir" = "$browseRoot" ]; then
        printf "%s\n" \
    else
        root=$(printf "%s\n" "$browseRoot" | sed 's/\\/\n/g')
        printf "%s\n" $currentDir | awk -v root=$root '{n=split(root,tab,"\\");
m=split($1,tab2,"\\"); for(i=n;i<=m;i++){final=final"\\"tab2[i]}}print final;' \
    fi
}
```

c) cd

Permet de se déplacer dans l'archive, soit en entrant dans un sous-répertoire du répertoire courant, soit en retournant dans le répertoire parent ("..") du répertoire courant. Par ailleurs l'utilisateur peut aussi entrer une destination partant de la racine pour naviguer sans se soucier des contraintes précédentes.

```

function browse-cd() {
    dirDestination=$1
    test=$(printf "%s\n" $dirDestination | cut -c1 | sed 's/\\\\\\\\\\\\\\\\/g')
    if [[ ! -z $test ]]; then
        test=$(egrep "^\directory.*$test$" archives/exemple | cut -d" " -f2)
    else
        test="0"
    fi
    if [[ "$dirDestination" = '\' ]]; then
        currentDir=$browseRoot
    elif [[ "$dirDestination" = ".." ]]; then
        if [[ "$currentDir" != "$browseRoot" ]]; then
            currentDir=$(printf "%s\n" "$currentDir" | sed 's/\(.*)\\\.*/$1/')
            test=$currentDir\''
            if [[ "$test" = "$browseRoot" ]]; then
                currentDir=$test
            fi
        fi
    elif [[ "$test" = "$browseRoot" ]]; then
        dir=$(printf "%s\n" "$dirDestination" | sed 's/\\\\\\\\\\\\\\\\/g')
        dir=$(egrep "^\directory.*$dir$" archives/$browseArchive | cut -d" " -f2)
        if [[ -z $dir ]]; then
            echo "pas de dossier $dirDestination connu"
        else
            currentDir=$dir
        fi
    else
        if [[ "$currentDir" = "$browseRoot" ]]; then
            dir=$currentDir$dirDestination
        else
            dir=$currentDir'\'$dirDestination
        fi
        dir=$(printf "%s\n" "$dir" | sed 's/\\\\\\\\\\\\\\\\/g')
        dir=$(egrep "^\directory.*$dir$" archives/$browseArchive | cut -d" " -f2)
        if [[ -z $dir ]]; then
            echo "pas de dossier $dirDestination connu"
        else
            currentDir=$dir
        fi
    fi
}

```

d) ls

L'objectif de cette commande est de lister les répertoires et les fichiers présents dans le répertoire courant. Pour cela nous devons déterminer quels sont les arguments donnés par l'utilisateur pour choisir un mode de traitement et d'affichage adapté. Nous séparons donc chaque cas par l'utilisation d'un case. Ensuite nous faisons appel à la fonction IsAll() pour déterminer le chemin du répertoire à étudier (car l'utilisateur peut spécifier aucun répertoire, ce qui par défaut utilise le répertoire courant, ou bien il peut aussi utiliser un sous-répertoire du répertoire courant ou bien un chemin partant de la racine).

Enfin nous utilisons awk pour chercher dans l'archive le contenu du dossier trouvé précédemment et réaliser les traitements nécessaires pour l'affichage (propre à chaque case).

```

function lsAll() {
    test=$(printf "%s\n" $1 | cut -c1 | sed 's/\\\\\\\\/g')
    if [[ ! -z $test ]]; then
        test=$(egrep "^\directory.*$test$" archives/exemple | cut -d" " -f2)
    else
        test="0"
    fi
    if [[ -z $1 ]]; then
        dir=$currentDir
    elif [[ "$test" = "$browseRoot" ]]; then
        dir=$browseRoot$(printf "%s\n" $1 | cut -c2-)
    else
        if [[ "$currentDir" = "$browseRoot" ]]; then
            dir=$currentDir$1
        else
            dir=$currentDir'\'$1
        fi
    fi
    dir=$(printf "%s\n" "$dir" | sed 's/\\\\\\\\/g')
    printf "%s\n" $dir
}

function browse_ls() {
    case $1 in
        -l )
            dir=$(lsAll $2)
            awk -v dir=$dir 'BEGIN{strDir="directory "dir}{if($0==strDir){getline; while($1!="@") {split($1,tab,"");if(tab[1]!="."){print $2" "$3" "$1} getline}}}' archives/$browseArchive
            ;;
        -a )
            dir=$(lsAll $2)
            awk -v dir=$dir 'BEGIN{strDir="directory "dir}{if($0==strDir){getline; while($1!="@") {ls=ls" "$1; split($2,tab2,"");if(tab2[1]=="d"){ls=ls"\\"}else if(tab2[1]=="-" && (tab2[4]=="x" || tab2[7]=="x" || tab2[10]=="x")){ls=ls"*"} getline} print ls}}}' archives/$browseArchive
            ;;
        -la | -al )
            dir=$(lsAll $2)
            awk -v dir=$dir 'BEGIN{strDir="directory "dir}{if($0==strDir){getline; while($1!="@") {print $2" "$3" "$1; getline}}}'' archives/$browseArchive
            ;;
        * )
            dir=$(lsAll $1)
            awk -v dir=$dir 'BEGIN{strDir="directory "dir}{if($0==strDir){getline; while($1!="@") {split($1,tab,"");if(tab[1]!="."){ls=ls" \"$1" split($2,tab2,"");if(tab2[1]=="d") {ls=ls"\\"}else if(tab2[1]=="-" && (tab2[4]=="x" || tab2[7]=="x" || tab2[10]=="x")){ls=ls"*"} getline} print ls}}}' archives/$browseArchive
            ;;
    esac
}

```

4. extract()

Le mode extract() a pour action de créer dans le répertoire courant toute l'arborescence de répertoires et les fichiers contenus une archive donnée. De ce fait, le serveur restaure l'arborescence et les fichiers dans le répertoire courant de la machine cliente. Par ailleurs, les droits relatifs aux dossiers, sous-dossiers et à chaque fichier sont conservés. Le serveur extrait le contenu d'une archive dont le nom est renseigné en argument. Tout d'abord, le nom de l'archive correspondant à l'argument entré par l'utilisateur est récupéré. Si il n'y a aucun argument, alors la fonction retourne un message d'erreur. Sinon, la fonction vérifie ensuite que cet argument correspond à une archive qui existe dans le serveur. Elle cherche des occurrences de cette archive avec la fonction grep. Par la suite, le chemin pointant vers l'archive est récupéré et on crée les variables début et header qui prennent respectivement les valeurs du début du header et le nombre de lignes du header. Ces valeurs se situent à la première ligne de l'archive et la fonction cut est utilisée pour les extraire. Par la suite, on récupère l'archive dans un fichier temporaire dans lequel le traitement est effectué. Ceci a pour but d'éviter d'effectuer des manipulations sur un fichier en lecture. On s'assure que les '\' de l'archive sont préservées en utilisant la fonction sed.

```

function commande-extract() {
    nomArchive=$1 #Argument
    if [ -z $nomArchive ] #Si aucun argument
    then
        echo "Erreur, aucune archive donnée en argument"
        return
    elif [ -n $nomArchive ]
    then #Si argument, vérifier que l'archive existe bien dans le serveur
        trouve=$(ls archives | grep -c $nomArchive) # Chercher occurrence de l'archive dans le serveur
        if [ $trouve -eq 0 ]
        then
            echo "L'archive n'existe pas dans le serveur"
            return
        else
            chemin=archives/$nomArchive #Path de l'archive
            debut=$((head -1 $chemin | cut -d":" -f1)) #Début du header
            header=$((head -1 $chemin | cut -d":" -f2)) # nombre de lignes du header
            cat $chemin > tmp_extract/archive_tmp #Récupération de l'archive dans un fichier
temporaire
            sed -i 's/\\\\\\\\/g' tmp_extract/archive_tmp # Je remplace les \ par les /
            i=0
        fi
    fi
}

```

De plus, on initialise un compteur i à 0 qui est incrémenté de 1 jusqu'à atteindre le nombre de lignes du header. Ce compteur permet ainsi de traiter uniquement l'arborescence et donc de créer dans un premier temps l'ensemble des dossiers, sous-dossiers et fichiers avant de transférer le contenu du body dans les fichiers adéquats. Le compteur est utilisé dans une boucle while qui effectue une lecture ligne par ligne du fichier temporaire. Si le compteur est plus petit que le numéro de ligne du début du header, on ignore et on continue. Si le compteur est égal à la dernière ligne du header, on sort de la boucle. Chaque ligne est séparée par des arguments délimités par un espace. On paramètre cela avec la fonction set. Par ailleurs, si la ligne en lecture est égale au @, on ignore. Par contre, si la ligne commence par le mot directory, la fonction awk est invoquée afin de saisir le 2ème champ qui correspond à l'arborescence qui est stockée ensuite dans la variable arbo_doss. Grâce à cela, il est possible de créer cette arborescence avec la fonction mkdir et l'option -parents qui permettra de créer récursivement des dossiers.

```

while read ligne #Lecture ligne par ligne du fichier temporaire avec que le contenu du header
do
    i=$((i+1)) #Compteur pour savoir si on est encore dans le header ou non
    if [ "$i" -lt "$debut" ]; then # Si le compteur est plus petit que la ligne de
debut du header, skip
        continue
    fi
    if [ "$i" -eq "$header" ]; then # Si le compteur est égale à la dernière ligne du
header
        break #Je quitte la boucle
    fi
    set $ligne # On prend les arguments de chaque ligne
    if [[ "$ligne" == "@*" ]]; then #Si la ligne est égale à @, on skip
        continue
    fi
    if [[ "$ligne" == "directory"* ]]; then # Vrai si la ligne commence par directory
        then
            arbo_doss=$(echo $ligne | awk '{print $2}') # Si ça commence par directory, je
prend le field 2 qui correspond à l'arbo
            mkdir -p tmp_extract/$arbo_doss # Création avec mkdir l'arbo avec l'option -
parents
    fi
done

```

Sinon, si la ligne ne commence pas par directory, dans ce cas, il s'agit d'un fichier ou d'un sous-dossier. Nous récupérons les droits que nous stockons dans une variable rights avec la fonction awk pour saisir le 2ème champ correspondant aux droits. De manière analogue, nous faisons de même avec le nom du fichier ou du sous-dossier. Pour distinguer fichier et sous-dossier, nous regardons si les droits commencent par d ou par -. Si il s'agit d'un dossier, la fonction mkdir est utilisée et l'option -m permet d'attribuer à chaque dossier les droits par défaut 755 en notation octale que nous stockons de manière récursive dans \$arbo_doss/\$name. Ensuite, si c'est un fichier à traiter, le script crée d'abord un fichier vide avec la fonction touch et par la suite, une conversion notation symbolique vers notation octale est effectuée peu importe la combinaison de droits avec l'utilisation de sed et d'expressions régulières. Cette conversion est stockée dans la variable roctale puis la commande chmod est utilisée pour attribuer à chaque fichier les droits respectifs.

```

        elif [[ ! "$ligne" =~ "directory"* ]] # Sinon, si ce n'est pas un directory
            then
                rights=$(echo $ligne | awk '{print $2}') # Récupération des droits
                name=$(echo $ligne | awk '{print $1}') # Récupération des noms
                if [[ "$rights" == "d"* ]] # Si les droits commencent par un d alors je sais que
c'est un sous-dossier
                    then
                        mkdir -m 755 tmp_extract/$arbo_doss/$name # Je fais un mkdir avec chmod 755
pour un sous dossier dans le repertoire main
                elif [[ "$rights" == "-"* ]]
                    then
                        touch tmp_extract/$arbo_doss/$name # Création d'un fichier vide dans le
repertoire main avec le nom enregistré
                        roctale=$(echo "$rights" | sed 's/.\\((.....)\\.*/\\1/
h;y/rwsxtST1L-/IIIII00000/;x;s/..\\(.)..\\(.)..\\(.)//;\\1\\2\\3/
y/sStT1Lx-/IIIIIOO/;G
s/\\n\\(.*)/\\1;000000I10I020II3I004IOI5II06III7/;:k
s/|\\(...\\)(.*;.*\\1\\(.)\\)/\\3|\\2/;tk
s/^0*\\(..\\)\\.*\\1\\q'/ # Conversion notation symbolique vers octale
chmod $roctale tmp_extract/$arbo_doss/$name # J'applique les droits respectifs
à chaque fichier

```

La taille du fichier est récupérée et est stockée dans la variable taille. Si cette taille est différente de 0, alors le numéro de la ligne où commence le contenu du fichier est récupéré et il en est de même pour l'occupation totale en nombre de lignes du contenu. Bien-sûr, si il s'agit d'un fichier vide, nous ignorons le fichier car il n'y a aucun contenu à transférer dans ce cas. Par la suite, on stocke dans un pipe le fichier temporaire et on récupère l'intervalle contenant les données avec la commande sed -n “debut_data”, ‘nbr_lignes_data’ p” qui affiche uniquement un intervalle de texte. Enfin, on compresse tout ceci dans une archive tar.xz avec la commande TAR qui permet d'envoyer l'arborescence créée précédemment au client (car l'on ne peut pas réaliser de cat sur un répertoire).

Nous utilisons netcat en mode listening pour les mêmes raisons que pour toutes les fonctions précédentes : seul l'utilisateur à besoin de renseigner une adresse, et cela permet de respecter un fonctionnement “type” serveur / client.

```

taille=$(echo "$ligne" | awk '{print $3}') #Si taille differente de zero, recuperation du contenu
if [ "$taille" -ne 0 ] #Sinon, je passe la ligne (Aucun contenu a recuperer)
then
    bodycommence=$((header-1+$(echo "$ligne" | cut -d" " -f4))) #Recuperation ou
le contenu commence
    bodyetendre=$(echo "$ligne" | awk '{print $5}')
    if [ 0 -eq "$bodyetendre" ]; then #Si fichier vide (Aucune info
complementaire, on skip on cherche pas le contenu)
        continue
    fi
    bodyetendre=$((bodycommence-1+bodyetendre)) #Recuperer jusqu'où le contenu
s'estendre
    contenu=$(cat "tmp_extract/archive_tmp" | sed -n
"$bodycommence,$bodyetendre p") #Contenu de chaque fichier a mettre dans le fichier en question
    echo "$contenu" > "tmp_extract/$arbo_doss/$name" #Transfert du contenu dans
le fichier respectif
    fi
    fi
    fi
done < tmp_extract/archive_tmp #Lecture
fi
fi

cd tmp_extract
rm archive_tmp
tar Jcvf send.tar.xz *
cat send.tar.xz | nc -l -p 8081
cd ..
rm -rf tmp_extract/*

echo "Extraction terminée"
}

```

Analyse

Cette dernière partie est consacrée à l'analyse des choix qui ont conduit à l'aboutissement du projet. Nous proposons ici de critiquer nos réflexions et d'évaluer les différences entre ce que nous comptions faire et ce que nous avons fait. Pour en savoir plus concernant la structure envisagée du programme, veuillez vous référer à notre premier rapport. Le codage des idées finales est le résultat du script bash. L'implémentation de la fonction list() a été effectuée en concordance avec nos idées en début de projet. Pas de différences à noter. Il en est de même pour la fonction browse(), cat() et touch(). Concernant la fonction extract(), nous avions pensé à faire en sorte que le serveur puisse transmettre l'archive au client pour que ce dernier puisse ensuite en extraire le contenu. Cependant, nous avons réussi à imiter le comportement normal d'un serveur recevant une requête. Dans la version finale du projet, c'est bien le serveur et non le client qui se charge d'extraire le contenu d'une archive pour ensuite transférer les différents fichiers et dossiers au client au travers d'un fichier au format.zip. C'est un fonctionnement similaire à Google Drive, lors du téléchargement l'on obtient aussi une archive zip. Néanmoins, le pseudo-code établi a été plus ou moins respecté. Par la suite, nous avons conclu à l'utilisation de sed, awk, grep et chmod et ces fonctions ont bien été implémentées. Par ailleurs, des réflexions supplémentaires ont été apportées après la rédaction du premier rapport, comme la possibilité d'utiliser la commande TAR par exemple lors des transferts client serveur. De plus, le script vsh.sh client n'a pas été abordé durant le premier rapport mais son implémentation a été un succès. Néanmoins, nous n'avons pas réussi à implémenter correctement les fonctions mkdir et rm du mode browse. Ces dernières sont donc manquantes mais nous pourrions très bien continuer à étudier la question dans de prochains travaux. La prochaine étape logique serait de permettre à plusieurs utilisateurs de communiquer avec le serveur en même temps

Conclusion

Pour conclure, nous avons trouvé que ce projet a été très formateur. Nous avons appris la programmation en bash et plus généralement au niveau de la gestion de projet, en construisant le projet de A à Z pas à pas, en passant par une phase de réflexion, de recherche de solutions, d'expérimentation, et la mise en place de la solution retenue.