Architecture TSH

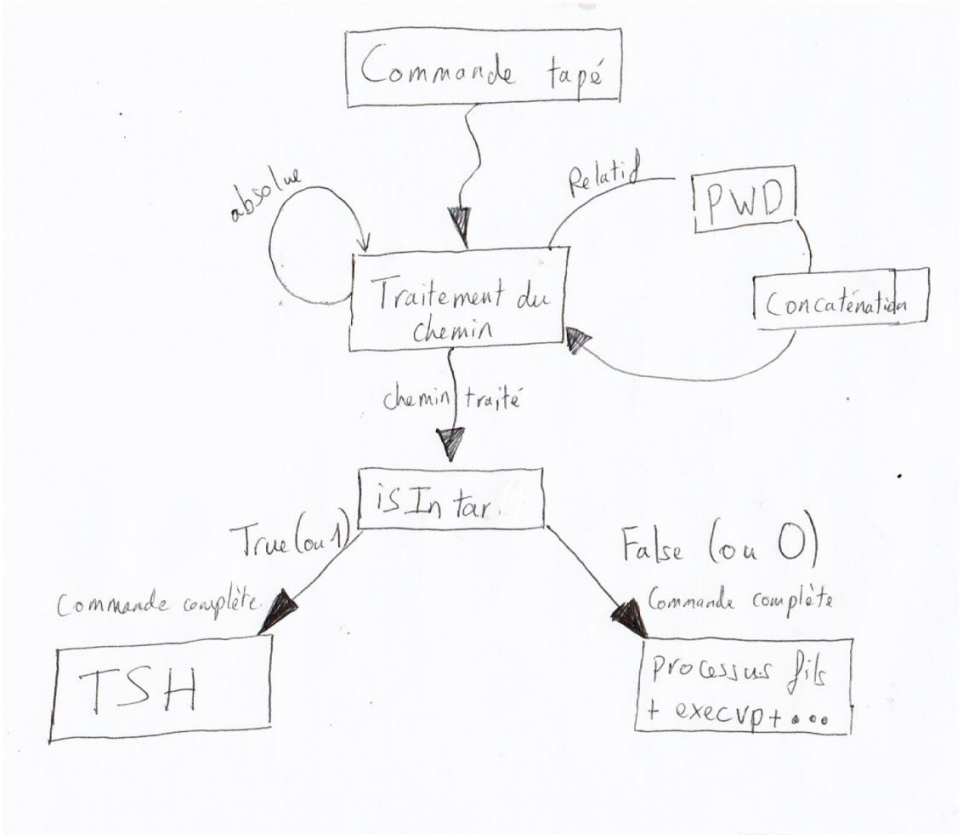
Introduction : Ce document sert à expliquer la démarche prise pour l'implémentation du shell pour les TarBall, TSH. Il faut savoir que nous considérons pour l'instant les options des commandes ainsi que les redirections comme étant peu prioritaire pour l'instant.

I. Comment savoir si la commande entre dans un tarball ou non ?

Pour pouvoir résoudre ce problème, nous avons choisi d'utiliser la variable d'environnement PWD comme contenant pour le chemin courant dans les tar. Ainsi si le répertoire courant est contenu dans un tar, la variable PWD contiendra dans son contenu un nom de fichier avec l'extension '.tar'.

Pour savoir si une commande entre dans un tar, il nous suffit ainsi de concaténer le chemin relatif indiqué avec une copie de PWD pour voir si un fichier '.tar' apparaît ou non.
Dans le cas d'un chemin absolu, il suffit de faire la vérification directement dessus sans prendre en compte de la variable d'environnement PWD.
(la concaténation/traitement du chemin de la commande aura par exemple le comportement suivant :
soit PWD = "/home/Bureau" et path = "a/b/c.tar/d/toto",
le résultat donnera : "/home/Bureau/a/b/c.tar/toto")

Selon le résultat, la commande sera soit traitée par l'un de nos programmes, soit envoyée à un autre processus fils pour qu'il exécute la commande avec execvp (chose que l'on doit encore implémenter).



II. Stratégie pour l'implémentation des commandes

Dans un premier temps, nous avons décidé de ne pas nous attaquer aux commandes (ce choix a été fait après le premier sprint, il y a donc des fichiers de commande mais les implémentations à l'intérieur ne sont pas pertinentes et seront probablement effacés dans le futur), mais plutôt de programmer des fonctions 'bas niveau' qui nous permettront de manipuler les tarballs de manière plus simple et ainsi de fabriquer les commandes à partir de ces fonctions. Si le sprint de cette semaine c'est bien passé, nous devrions avoir les fonctions suivantes :

-ouverture d'une archive

[int openArchive (char * path, int flags)]

-récupération du header d'un fichier

[int readHeader (int fd, struct posix_header buf)]

-récupération du contenu d'un fichier

[char * getContent (int fd, struct posix_header * h)]

-une fonction passant le contenu d'un fichier

[void passContent (int fd, struct posix_header * h)]

-récupération du header d'un fichier et passe directement au prochain fichier.

Vous avez sûrement remarqué que pour l'instant, nous n'avons que des fonctions de lecture et aucune d'écriture,c'est normal, cependant on devrait normalement être capable de faire les commandes cd, pwd, ls, cat.

Il faudra encore que l'on réfléchisse à comment implémenter mkdir,rmdir,mv,cp,rm.

Toutes ces fonctions seront normalement (si le sprint s'est bien passé) contenu dans un fichier qui nous servira de bibliothèque pour la suite.

Nous testons les fonctions et commandes à la main sans utiliser le shell que l'on doit implémenter plus tard, il faudra évidemment dans le futur faire un shell qui puisse lancer ces commandes directement mais nous avons jugés qu'il ne s'agissait pas encore de la priorité.

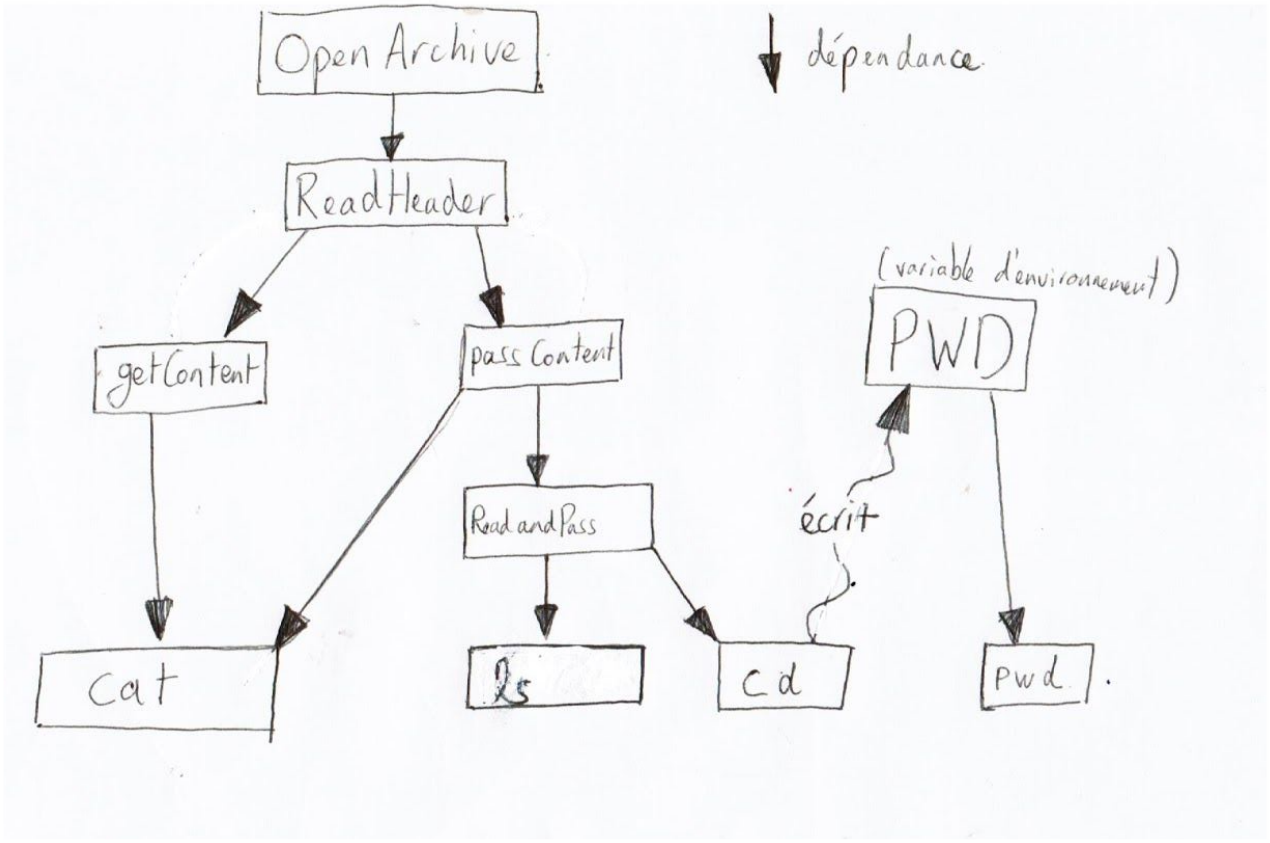
III. Fonctionnement générale d'une commande dans le tar

Une commande fonctionnera de manière générale de cette manière :

1) Reçois le chemin complet demandé (ce chemin contiendra forcément un .tar)
2) Découpe le chemin en deux parties: la première est la référence absolue vers l'archive.tar et la seconde est le reste du chemin après l'archive.tar
3) Ouvre l'archive grâce à la première partie du chemin
4) Effectue les opérations de la commande avec comme information la deuxième partie du chemin
5) Ferme l'archive après opération.

Le chemin sera bien évidemment construit à partir d'une concaténation du PWD et du chemin relatif donné par l'utilisateur ou juste du chemin absolu de l'utilisateur. (Cf partie I)

IV. Diagramme de composition pour les fonctions



V. Idées des implémentations pour le future du projets.

cp :

-Dans le cas où les deux chemins sont dans un tar, on a l'idée de lire les octets du contenu du fichier à l'aide des fonctions readHeader et getContent précédemment implémenter. Ensuite avec l'aide d'une fonction qui mettra le curseur du read à la fin du fichier,créer un header avec le bon nom de fichier et écrire tous les octets que l'on a récupéré juste après le header.

-Dans le cas où le chemin du fichier à copier est dans le tar mais pas la destination, il faudra copier le contenu du fichier à copier avec readHeader et getContent et ensuite ouvrir un nouveau fichier dans lequel nous allons recopier le contenu.

-Dans le cas où le chemin du fichier à copier est en dehors du tar mais que la destination est dans le tar, il faudra copier le contenu du fichier, mettre le curseur à la fin du fichier, créer un header de 512 octets et écrire ensuite le contenu du fichier.

Nous n'avons aucune idée de comment créer nous même le header pour l'instant. Si vous avez des suggestions de comment l'implémenter, nous serons ravi de les avoir.

mv:

Si les deux chemins sont dans le tar, il suffira juste de modifier le nom contenu dans le header pour qu'il corresponde au nouveau nom (après vérification qu'il soit possible d'être dans le fichier et qu'il n'y ait pas d'autre fichier de même nom)

Si le fichier à déplacer est en dehors du tar, il faut d'abord supprimer (après avoir copié son header et contenu) le fichier et le recréer dans le tar à la fin (créer un header et y écrire son contenu ensuite).

Si le fichier à déplacer est dans le tar, il faudra copier son contenu (avec readHeader et getContent) et ouvrir un nouveau fichier dans la destination et écrire le contenu dans ce nouveau fichier.

mkdir:

Pour mkdir, on s'est dit qu'il suffisait de créer un header dans le tar avec le nom correspondant (après avoir vérifié si il était possible de le créer à l'emplacement voulu), un typeflag = '5' (qui correspond normalement aux répertoire) et enfin avec un contenu = 0.

rm:

Pour rm, on s'est dit qu'il fallait faire une fonction qui retourne la taille totale de l'archive après le fichier à supprimer,puis récupérer tous les octets des fichiers du tar qui suivent le fichier à supprimer et écraser ce qu'on a récupérer par dessus le fichier à supprimer. Il faudra ensuite mettre à 0 tous les octets après l'écrasement.

rmdir: Pour rmdir, il faudra d'abord vérifier qu'il n'y a aucun fichier dans le répertoire à supprimer dans un premier temps. Puis si il est possible de le supprimer, utiliser rm pour supprimer le header du répertoire.

Fonction utile que l'on devra programmer :

- Fonction qui mettra le curseur du read/write à la fin du tar (on considère que l'on est à la fin d'un tar si on tombe sur un header avec le nom = "")
- Fonction qui créera le posix header en contenant les bonnes informations. (on ne sait pas encore comment le faire)
- Fonction qui écrira dans le tar le posix header et le contenu (en respectant les blocs de 512)
- Fonction qui vérifiera que les chemins vers des répertoires
- Fonction qui récupère la taille de l'archive après un fichier
- Fonction qui récupère tous les octets après un fichier
- Fonction qui vérifie qu'il n'y a pas de fichier dans un certain repertoire