

## PROJET : Archivage « ctar » – à réaliser en binôme – Durée approximative ~ 12h00min.

A l'aide des supports de cours et des mémentos et des exercices achevés en TP et du manuel Linux réalisez :

(Tout code ou implémentation compilant ou non sera étudié, toute fois le barème sera adapté)

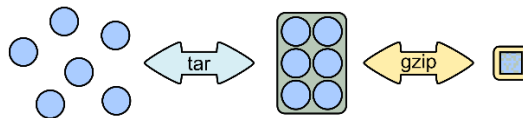
Le présent sujet de projet comporte 3 pages

### I – Synopsis

L'objectif de ce projet est de réaliser *-dans une proportion simplifiée-* l'implémentation d'un générateur/extracteur d'archive **.tar** similaire à la commande éponyme **tar**. Ce dernier doit pouvoir manipuler une archive de type **.tar** respectant la norme GNU POSIX 1003.1-1988 dite « ustar ».

Une archive tar est fichier résultant de la volonté de regrouper un ensemble de fichiers en un seul. Le nom de la commande est un raccourci pour « tape archive » ce dernier est dit non compressé mais peut le devenir grâce à l'intervention en aval de la commande **gzip** par exemple.

Nous retrouvons ainsi la schématisation usuelle d'un ensemble de fichiers aboutissant à une archive **.tar.gz** :



### Exemple:

**Prompt>** `tar -l my_archive.tar` **Prompt>** `tar -x my_archive.tar`

**Prompt>** `tar -c my_archive.tar file1 file2 file3`


### II – Démarche

En préambule il faut absolument bien lire la page de manuel de tar(5), et éventuellement compléter cette lecture par d'autres documents. Il est recommandé d'utiliser le format GNU tar archives. Attention il est important de faire attention à la description des différents champs et de leur définition.

Il faut faire attention au fait qu'une archive tar est une succession de blocs de **512 octets**, qu'il est préférable de lire l'un après l'autre dans leur ensemble. Ainsi dans le cadre du parcours d'une archive il sera nécessaire de lire de façon répétée les entêtes correspondant au format étudié pour chaque item composant cette dernière.

Concrètement une instruction **read()** sera à réitérer tout au long du fichier **.tar** dont la destination est une structure :

```
struct header_tar file_header;
read(fd, &file_header, 512);
```

 **nombre de fichiers présents dans l'archive**

Une fois la structure peuplée par les données du fichier il est alors possible d'opérer l'action sollicitée.

Il est à noter qu'entre deux fichiers se trouve le contenu du fichier succédant à l'entrée précédemment lue. La condition de fin d'archive est quant-à elle opérée par deux blocs de 512 octets remplis de blanc binaire (Soit 2x512 octets à 0).

Ainsi dans le cadre d'une extraction on distinguera trois phases distinctes :

1. La récupération de l'entête correspondant au fichier.
2. L'analyse des données de l'entête.
3. L'écriture du fichier sur le disque.

### III – Résultats attendus

Le livrable attendu pour ce projet se résume en un code source compilable et exécutable sous linux répondant d'une part aux fonctions métiers suivantes :

- **FM01** – Le binaire est capable de lister les éléments d'une archive.
- **FM02** – Le binaire est d'extraire l'intégralité d'une archive passée en paramètre.
- **FM03** – L'application est capable de générer une archive.

D'autres fonctionnalités supplémentaires peuvent être prises en compte :

- **FMO01** – La prise en charge de la compression d'une archive tar (via la librairie zlib).
- **FMO02** – La prise en charge de la décompression d'une archive tar.gz (gzip).
- **FMO03** – La réalisation d'une interface en mode console « tui » via la librairie ncurses.

Les paramètres disponibles devront être de la sorte :

```
-l, --list      ARCHIVE_FILE
-e, --extract   ARCHIVE_FILE
-c, --create    ARCHIVE_FILE
-d, --directory DIRECTORY_TO_PROCESS
-z, --compress

-v, --verbose : enable *verbose* mode
-h, --help    : display this help
```

Concernant les exigences techniques attendues, vous devez respecter les contraintes suivantes :

- **CT01** – La compilation du projet doit se faire via un **Makefile**.
- **CT02** – La définition des structures doit se faire dans un fichier **typedef.h**.
- **CT03** – La définition des méthodes prototype (.h) & implémentation (.c) doit se faire de manière séparée autant que faire se peut.
- **CT04** – Le code produit doit être documenté.
- **CT05** – La récupération des paramètres doit se faire via les fonctions usuelles **getopt()** / **getoptlong()** .
- **CT06** – La gestion des erreurs doit se faire via « les mécanismes proposés par errno ».

D'autres contraintes techniques peuvent être prises en compte :

- **CTO01** – La documentation du code générée via l'utilitaire **doxygen**.
- **CTO02** – Le code est soumis à un contrôle de couverture via l'utilitaire **gcov**.
- **CTO03** – Une page de manuel Linux est rédigée pour détailler l'exécution du binaire.

#### **IV – Evaluation**

Ce projet est à réaliser en monôme ou en binôme (au choix) et donnera lieu à deux livrables :

- Le code source du projet répondant à la problématique exposée.
- Un rapport détaillant la logique implémentée (3 pages maximum).

La réalisation de cette commande simplifiée vise à mettre en œuvre l'ensemble des connaissances de programmation abordées au cours de ce module. La restitution du plus grand nombre de notions à travers le code produit vous permet de valider vos compétences.

Ainsi la réalisation de l'ensemble des fonctionnalités métiers **FM 1** à **3** ainsi que le respect des contraintes techniques vous assure une note supérieure à la moyenne signifiant l'acquis des connaissances. Toutefois la réalisation des fonctionnalités métiers et/ou optionnelles vous permettent d'augmenter votre note le cas échéant.

##### **IV.1 – Rapport**

Vous procéderez à la rédaction d'un mini-rapport de projet (3 pages maximum, format pdf).

Vous y détaillerez les éléments suivants :

- Choix de conception.
- Les difficultés rencontrées ainsi que leurs résolutions.
- Nombre d'heures passées sur les différentes étapes de ce projet :  
(Conception, codage, tests, rédaction du rapport) pour chaque une des personnes composant le binôme.

## V – Rappels

### Approche incrémentale du développement

Pour parvenir au résultat attendu, veuillez toujours appliquer une approche incrémentale en termes d'ajout de code/fonctionnalité, procédez par étape afin de ne pas avoir un code C trop complexe qui serait *in-fine* difficile à débbugger. Il est ainsi primordial de procéder selon une logique construite, réfléchie, tout en procédant régulièrement au test du code comprenant bien sur les différents cas d'erreur.

Par exemple: une approche incrémentale pour ce type d'exercice serait de procéder dans l'ordre :

1. La récupération des paramètres
2. La lecture d'une archive tar (lister les fichiers d'une archive)
3. L'implémentation d'un extracteur d'archive
4. L'implémentation d'un générateur d'archive
5. Implémentation des éventuels bonus

### Documentation

Pour obtenir des informations ou de la documentation ayez le réflexe d'utiliser les pages du manuel.

Par exemple:

- man 3 stat / man 2 open / man 2 readdir / man errno / man 5 tar
- Description et format -GNU tar- : [https://www.gnu.org/software/tar/manual/html\\_node/Standard.html](https://www.gnu.org/software/tar/manual/html_node/Standard.html)
- Librairie ncurses : <https://tldp.org/HOWTO/NCURSES-Programming-HOWTO/>

### Limitation d'usage

Vous pouvez vous servir de n'importe quelle librairie tierce pour réaliser l'implémentation de vos algorithmes et /ou structures dans la mesure où elle n'implémente pas directement une fonctionnalité métier.

Afin de simplifier l'implémentation du projet vous vous limiterez à l'extraction des répertoires et tous les fichiers seront à considérer comme des fichiers normaux.

### Gestion des erreurs

Afin d'avoir une gestion des erreurs la plus précise possible ayez le réflexe d'utiliser les codes retours **ERRNO** spécifiés dans les pages de manuel.

Par exemple:

- |                  |   |
|------------------|---|
| ➤ <b>EEXIST</b>  | File exists (POSIX.1)                       |
| ➤ <b>EFAULT</b>  | Bad address (POSIX.1)                       |
| ➤ <b>EISDIR</b>  | Is a directory (POSIX.1)                    |
| ➤ <b>ENOTDIR</b> | Not a directory (POSIX.1)                   |
| ➤ <b>ELOOP</b>   | Too many levels of symbolic links (POSIX.1) |