Université de Lorraine
Telecom Nancy 2eannée
(2016-2017)

Rapport de projet

Sujet : ptar – un extracteur d'archives tar durable et parallèle

Charlotte Devy
Guillaume Ruchot

Date de rendu : 15 décembre 2016
Module de Réseaux-Systèmes

Table des matières

Remerciements

Nous tenons à remercier tout particulièrement M. Lucas Nussbaum pour son aide tout au long de ce projet ainsi que pour les différents tests blancs qu'il a mis en place et qui nous ont permis de comprendre les résultats attendus.

Nous avons réalisé ce projet en nous aidant des sites webs suivants :
— https ://members.loria.fr/lnussbaum/RS/
— http ://stackoverflow.com/
— http ://www.zlib.net/manual.html

1 Introduction

Ce projet a consisté à concevoir et implémenter un extracteur d'archives tar durable et parallèle en langage C. Il nous permis de mettre en œuvre les connaissances acquises dans le module de Réseaux-Systèmes, notamment les méthodes liées à la gestion efficace des threads.

2 Choix de conception

Tout d'abord, il nous a fallu choisir comment notre programme s'exécutera dans sa globalité : c'est à dire si on extrait les fichiers au fur et à mesure qu'on lit les headers ou si on lit plusieurs fois le fichier tar (une pour l'initialisation et une fois pour l'extraction).

Nous avons choisi la seconde solution car elle nous permet de gérer le cas de l'affichage des éléments de l'archive sans pour autant avoir à les extraire. De plus, cette solution nous permet d'éviter de nombreux problèmes de synchronisation que l'on aurait pu avoir lors de l'utilisation des threads (par exemple si les threads finissent d'extraire un fichier avant que le header permettant d'extraire le fichier suivant ne soit lu).

Par la suite, nous avons cherché la meilleure méthode nous permettant d'extraire les fichiers. Nous avons choisi d'écrire une méthode spécifique pour l'extraction, facilement adaptable pour la future implémentation des threads. Nous avons également choisi de stocker tous les headers contenus dans le fichier tar dans des structures header_t (cette structure nous a été fournie pour le manuel de la commande tar) puis de stocker tous ces headers dans une pile. On peut alors passer en argument un pointeur vers la base de la pile à la fonction de l'extraction et ainsi implémenter les threads en ajoutant juste un mutex lors de l'extraction de la base de la pile.

Nous avions choisi de laisser de côté les parties 6 (zlib) et 7 (checksum) lors de la conception initiale car nous savions que l'implémentation de la compression n'impacterait pas le reste du programme et donc que l'on pourrait simplement l'ajouter à la fin.

3 Difficultés rencontrées

3.1 Étape 1 et 2 : Listing des fichiers et récupération des options

Les deux premières étapes ne nous ont pas posé de difficultés particulières autres que la compréhension des différents types d'archives tar (ustar et gnu).

3.2 Étape 3 : Extraction simple de l'archive

L'extraction simple des fichiers n'a pas posé de problèmes particuliers car nous nous étions renseignés auparavant sur la lecture et l'écriture de fichiers octet par octet et que nous étions guidés par l'énoncé du projet et par le manuel de la commande tar.

3.3 Étape 4 : Extraction avancée et listing détaillé de l'archive

L'extraction avancée des fichiers posait deux problèmes en particulier :

- Chaque fois que l'on extrait un fichier (dans un dossier autre que le dossier courant) la date de dernière modification de ce dossier est mise à jour. Ainsi, on ne pouvait pas se contenter de renseigner les informations de ces dossiers dès leur création et nous avons donc opté pour une relecture de la pile des headers après l'extraction complète de l'archive pour corriger les timestamp de tous les dossiers.
- La gestion des informations avancée des liens symboliques nous a forcé à changer les fonctions que l'on utilisait, ainsi que les structures de timestamp. De plus, nous avons rencontré une erreur sur l'une de nos machines de test car le type 'long' sur cette machine ne suffisait pas à stocker le timestamp. Ce problème a été résolu en utilisant le type 'long long' pour stocker cette information.

Concernant le listing détaillé de l'archive, nous avons choisi d'initialiser une chaîne de caractères représentant les permissions avec 10 tirets puisque le tiret représente l'absence de permission : 1 caractère pour le type de fichier (d pour un répertoire, l pour un lien symbolique, rien pour les fichiers), et 3 caractères pour chacun des 3 types d'utilisateurs (user, groups, others) correspondant aux permissions d'écrire (w), de lire (r) et d'exécuter (x).

3.4 Étape 5 : Gestion des threads

Cette étape ayant été prévue dès la conception, nous n'avons pas rencontré de problèmes particuliers lors de la gestion des threads.

3.5 Étape 6 : Décompression de l'archive (zlib)

Cette partie du programme est celle qui nous a posé le plus de problèmes. Le premier d'entre eux (et le plus difficile à régler) est qu'il y a très peu de documentation en ligne sur cette bibliothèque. Ceci a rendu sa compréhension difficile et a compliqué lourdement la résolution des problèmes que nous avons rencontrés lors de la phase d'implémentation.

Nous avions deux possibilités d'implémentation :
- décompresser l'archive au fur et à mesure de sa lecture
- décompresser entièrement l'archive dès le lancement du programme, la stocker dans une archive tar temporaire puis appliquer le reste de notre programme sur cette archive temporaire.

Nous avons adopté la seconde solution qui, bien que plus difficile à implémenter dans notre code, est plus proche de l'idée de conception initiale (plusieurs passes sur l'archive).

Nous avons aussi rencontrés des problèmes pour charger dynamiquement la librairie zlib car nous ne pouvions utiliser ni les chemins relatifs (cela empêcherait l'exécution du programme depuis l'extérieur du dossier), ni les chemins absolus (ils ne seraient pas portables sur différentes machines ou différents OS). Finalement, nous avons décider de laisser le système déterminer le chemin de zlib.

3.6 Étape 7 : Vérification la cohérence des données (checksum)

La principale difficulté a été de comprendre ce qui été attendu lors de cette étape. En effet, puisque la structure header_t contenait déjà un champs checksum, nous ne voyions pas l'intérêt de la recalculé.

4 Gestion de projet

Ci-dessous, le tableaux récapitulatif du nombre d'heures passées sur chaque partie du projet par chacun des membres :

	Charlotte DEVY	**Guillaume RUCHOT**
Conception	10h	10h
Codage	10h	15h
Tests	2h	5h
Rédaction du rapport	1h30	1h30
Total	23h30	31h30