Rapport de projet

PTAR – Un extracteur d'archives TAR durable et parallèle

Module Réseaux et Systèmes

Timothy Garwood, Valentina Zelaya

Année 2016-2017

Table des matières

Remerciements

L'objectif de ce projet était celui de réaliser un extracteur d'archives TAR qui garantirait une écriture *durable et parallèle* sur le disque des fichiers extraits.

Le programme a été codé en C et utilise les **appels systèmes** de l'API POSIX.

Nous avons réalisé ce projet en nous aidant des sites web, des outils suivants et des conseils des personnes suivantes :

Sites Web

- https ://www.gnu.org/software/libc/manual/html_mono/libc.html ↪ Pour consulter les spécification des appels système utilisées dans notre programme
- http ://stackoverflow.com/ ↪ Pour différentes questions, concernant par exemple l'utilisation de getopt() et les arguments obligatoires, les modes pour la création de fichiers, changer le timestamp d'un dossier quand on n'arrivait pas à le faire (cf. 'Difficultés Rencontrés').

Outils

- `gdb`
- `vagrant`
- `valgrind` : pour une bonne gestion de la mémoire
- `Electric Fence` : pour nous assurer de détecter des erreurs de gestion de mémoire que valgrind ne signalerait pas.

Conseils

- M. Nussbaum, pour le déboggage de l'utilisation de la mémoire avec Electric Fence
- Quentin Tardivon, Joris Vigneron et Graziella Husson, qui nous ont signalé notre erreur dans la modification des timestamps des dossiers et symlinks.

1 Difficultés rencontrées et solutions

Grâce aux test automatiques, on a très vite compris que ce projet était un projet difficile. En effet, notre programme devait implémenter les fonctionnalités démandées tout en se conformant aux formats attendus dans les tests, et ce avec une très bonne gestion de la mémoire. Nous avons donc essayé de profiter au maximum des tests blancs pour répérer les erreurs dans notre code et nous présenterons par la suite les difficultés que nous avons rencontrées.

1.1 Stockage des données des fichiers

Une de nos premiers erreurs fut celle de vouloir stocker les données du fichier dans une structure de type liste chaînée lors du premier parcours de l'archive tar, qui visait à lister son contenu. En effet, ceci est très lourd, en mémoire et en temps, car on peut se voir stocker énormément de données dans le tas et provoquer une `Segmentation Fault` puisqu'on sort de la zone réservée à l'écriture.

1.2 Saut des données avec `lseek()`

On a remarqué grâce aux tests que lorsque l'on sautait les données d'un fichier avec un appel à `lseek()` pour ne pas les stocker, on avait oublié de traiter un cas particulier : celui où la taille du fichier était un multiple de 512. On sautait alors un bloc de 512 octets en trop. On a donc utilisé un modulo pour détecter ces cas particulier et nous avonc employé cette technique par la suite (dans le listing détaillé et l'extraction).

1.3 Timestamps et droits d'utilisateur

Lors de l'extraction il était demandé de restituer la date de modification mais nous n'arrivions pas à modifier le timestamp des dossiers ni des liens symboliques. Ce fut peut-être une des étapes les plus consommatrice en temps et en réflexion du simple fait que on n'a pas réussi à trouver sur internet à ce problème. Néanmoins, en discutant avec nos camarades mentionnés dans la section remerciements nous avons compris que même après avoir correctement restitué la date de modification d'un dossier, l'écriture de fichiers dans ce dossier modifiait automatiquement cette dernière. Nous avons donc fait un deuxième parcours des fichiers, dossiers et liens extraits pour effectuer les modifications du timestamp.

En ce qui concerne les liens symboliques, nous avons utilisé `lutimes()`.

1.4 Utilisation de la mémoire

Lors de l'avant dernier test, nous avons obtenus de très nombreux segmentation faults qui étaient dues à une mauvaise allocation de mémoire, qui n'était pas détectable dans les tests que nous avions réalisés. Étant dans l'incapacité de reproduire ces Segmentation Faults nous avons contacté M.Nussbaum qui nous a dirigé vers l'outil Electric Fence. Cet outil permet de détecter beaucoup de problèmes de gestion de mémoire en causant des segmentation faults explicites. Avec la combinaison de Electric Fence, Valgrind, et GDB, nous sommes parvenus à débarasser notre code de mauvaise gestion de mémoire.

1.5 Threads

Les difficultés de la manipulation de threads résidaient dans la limitation du nombre de ces derniers à un paramètre d'execution et dans la gestion de la mémoire partagée par ces threads. Pour surmonter la première, nous avons utilisé un sémaphore permettant de commencer l'extraction d'un fichier lorsqu'un thread se libérait. Pour la seconde, nous avons utilisé un ensemble de tableaux contenant les différentes données utiles aux threads, ainsi qu'un tableau d'indices, et un tableau de mutexs : en vérouillant un indice, pour le réserver à un thread, on vérouillait par conséquent toutes les ressources associées à cet indice, pour les réserver au thread.

1.6 Etapes 6 et 7

Par manque sévère de temps vers la fin du semestre, avec des rendus de TP et de projets, des partiels, un MOOC et une Coding Week du 12 au 16 décembre, nous n'avons pas pu implémenter de façon satisfaisante (c'est-à-dire que nos fonctions ne passaient pas notre propre batterie de tests).

2 Temps consacré au projet

Étape	Timothy Garwood	Valentina Zelaya
Étape 1	4h	4h
Étape 2	0.5h	2.5h
Étape 3	5.5h	2.5h
Étape 4	8h	7h
Étape 5	28h	3h
Étape 6	0h	1h
Étape 7	0h	2h
Rédaction du rapport	2h	4h
TOTAL	**48h**	**26h**