ptar – un extracteur d'archives tar *durable* et parallèle

RS : Réseaux et Systèmes
Deuxième année

L'objectif de ce projet est de réaliser un extracteur d'archives *tar* garantissant l'écriture sur le disque des fichiers extraits. Afin d'assurer des performances raisonnables, l'extracteur utilisera plusieurs *threads* pour paralléliser les écritures sur le disque dur.

1 Logistique

1.1 Comment travailler

Apprendre à travailler en groupe fait partie des objectifs pédagogiques de ce projet. Il vous est donc demandé de travailler en binôme. Vous pouvez travailler par groupe de trois si vous le souhaitez (mais sachez que nous tiendrons compte de la taille des groupes lors de la notation – regardez du côté des extensions pour compenser). Il est **interdit** de travailler seul. Un ingénieur travaille rarement seul.

La suite du sujet contient une stratégie possible que vous êtes libre de suivre, ou non.

Il est **indispensable** que votre projet fonctionne sous linux.

1.2 Évaluation de ce projet

Rapport. Vous devez rendre un mini-rapport de projet (5 pages maximum hors annexes et page de garde, format pdf). Vous y détaillerez vos choix de conception, les difficultés auxquelles vous avez été confronté, et comment vous les avez résolues. Vous indiquerez également le nombre d'heures passées sur les différentes étapes de ce projet (conception, codage, tests, rédaction du rapport) par membre du groupe.

Soutenances Des soutenances pourront être organisées. Vous devrez nous faire une démonstration de votre projet et être prêts à répondre à toutes les questions techniques sur le codage de l'application.

Section «Remerciements» du rapport. Votre rapport doit contenir quelque chose comme la mention suivante : «Nous avons réalisé ce projet sans aucune forme d'aide extérieure» ou bien «Nous avons réalisé ce projet en nous aidant des sites webs suivants (avec une liste de sites, et les informations que vous avez obtenu de chaque endroit)» ou encore «Nous avons réalisé ce projet avec l'aide de (nommer les personnes qui vous ont aidé, et indiquez ce qu'ils ont fait pour vous)». Si la liste est trop longue, vous pouvez la déplacer en annexes (pour ne pas empiéter sur vos 5 pages de rapports). Rien ne vous empêche de vous faire aider, mais il vous est demandé un minimum d'honnêteté académique en listant vos aides. Tout manquement à cette règle sera sanctionné comme il se doit.

La tricherie sera **sévèrement punie**. Par se faire aider, on entend : avoir une discussion sur le design du code, y compris sur des détails techniques et/ou les structures de données à mettre en œuvre. Par tricher, on entend : copier ou recopier du code ou encore lire le code de quelqu'un d'autre pour s'en inspirer. Nous testerons l'originalité de votre travail par rapport aux autres projets rendus [1], et il est difficile de défendre en soutenance un projet que l'on n'a pas écrit entièrement.

«Rendu» du projet. Vous devez utiliser un dépôt SVN sur la forge de l'ESIAL (`https://forge.esial.uhp-nancy.fr/`). Créez votre projet comme un sous-projet de *RS 2011* (dans *Projets 2A*). Votre projet doit être privé pour ne pas que les autres binômes puissent y accéder. L'identifiant de votre projet doit être de la forme `rs2011-login1-login2` (où `login1` et `login2` sont les deux logins des membres du binôme). Ajoutez *Lucas Nussbaum* (login : `lnussbau`) aux développeurs de votre projet.

Le projet sera récupéré directement sur votre dépôt SVN à la date de fin du projet. Dès votre binôme constitué et votre projet créé, vous devez envoyer un mail à `lucas.nussbaum@esial.uhp-nancy.fr` pour indiquer :

- vos noms, prénoms, logins ;
- l'identifiant de votre projet ;
- le chemin d'accès SVN à votre projet, utilisable par l'utilisateur `lnussbau` (de la forme `http://forge.esial.uhp-nancy.fr/svn/rs2011-login1-login2/`).

1. `http://theory.stanford.edu/~aiken/moss/`

Afin de faciliter l'évaluation, il est demandé que votre dépôt SVN contienne :
- Un ficher `AUTHORS` listant les noms, prénoms et login des membres du groupe (une personne par ligne) ;
- Un `Makefile` compilant votre projet en créant un fichier exécutable nommé `ptar` ;
- Un fichier `rapport.pdf` contenant votre rapport au format PDF.

Batterie de tests. Une partie de la note sera attribuée de manière semi-automatique par une batterie de tests. Vous perdrez des points si votre programme ne se comporte pas comme attendu. Pour le bon fonctionnement de ce processus (et l'attribution des points correspondants), votre programme doit renvoyer un code d'erreur de 0 en cas de succès et seulement dans ce cas. Un projet ne compilant pas correctement sera sanctionné par une note adéquate.

Voir `http://www.loria.fr/~lnussbau/rs2011.html` pour d'éventuelles informations complémentaires.

Vos questions éventuelles peuvent être adressées à `lucas.nussbaum@esial.uhp-nancy.fr`.

2 Description du projet

Le projet de cette année vise à développer un extracteur d'archives *tar* garantissant l'écriture sur le disque des fichiers extraits. Afin d'assurer des performances raisonnables, l'extracteur utilisera plusieurs *threads* pour paralléliser les écritures sur le disque dur.

2.1 Archives TAR

tar est à la fois un format de fichiers, et le nom de l'outil standard dans le monde Unix pour manipuler ces archives. Les fichiers *tar* sont utilisés pour rassembler un ensemble de fichiers (et de répertoires) dans un seul fichier (*tar* signifie « goudron »). Les archives *tar* ne sont pas compressées. On utilise en général *tar* avec un compresseur (*gzip*, *bzip2*, etc.) pour obtenir des archives compressées (`tgz = tar.gz = tar + compression gzip`).

Pour plus de détails sur *tar*, consultez :
- `http://en.wikipedia.org/wiki/Tar_(file_format)`
- La page de manuel `tar(5)` (à ne pas confondre avec `tar(1)`, qui décrit la commande `tar`) : `http://manpages.debian.net/cgi-bin/man.cgi?query=tar&sektion=5`

2.2 Durabilité des écritures

Dans les bases de données, la durabilité (*durability* en anglais) est la propriété qui garantit qu'une transaction a été enregistrée de manière à survivre à une panne. En pratique, cela nécessite de n'indiquer la transaction comme terminée qu'une fois que les données ont été écrites sur un support durable (disque dur par exemple). La plupart des outils usuels n'offrent pas cette propriété : lorsqu'une copie de fichiers avec `cp` se termine, les données sont peut-être encore en cours d'écriture sur le disque dur par le système d'exploitation. Forcer la durabilité pour toutes les actions sur les fichiers est possible, mais cela ralentirait très fortement le fonctionnement du système, puisqu'il est nécessaire d'attendre l'écriture effective des données sur le disque dur après chaque action. L'utilisation d'écritures durables est donc limitée aux logiciels qui le nécessitent vraiment, comme les gestionnaires de bases de données ou certains serveurs réseaux [2].

D'un point de vue pratique, une séquence `open(); write(); close();` ne garantit pas l'écriture des données sur le disque dur au moment du `close();`. Pour attendre l'écriture effective des données, il faut utiliser l'appel système `fsync()` : `open(); write(); fsync(); close();`.

2.3 Multiplexage des écritures avec des threads

L'utilisation de `fsync();` lors de l'écriture de chaque fichier limite très fortement les performances. Ainsi, pour une archive dont l'extraction prend 2 secondes avec la commande `tar`, l'extraction avec un

2. Par exemple, un serveur SMTP (email) ne répond qu'il *accepte* un mail qu'une fois qu'il l'a réellement écrit sur le disque dur. La RFC 2821 indique (section 6.1) : *When the receiver-SMTP accepts a piece of mail (by sending a "250 OK" message in response to DATA), it is accepting responsibility for delivering or relaying the message. It must take this responsibility seriously. It MUST NOT lose the message for frivolous reasons, such as because the host later crashes or because of a predictable resource shortage.*

tar durable prend 90 secondes. Afin de limiter ce problème, il est proposé d'utiliser des threads POSIX pour réaliser plusieurs écritures simultanément. Le système d'exploitation pourra ainsi réordonner les écritures sur le disque dur d'une manière plus efficace.

Sur l'archive mentionnée précédemment, utiliser des threads permet de diviser par trois le temps d'exécution.

Plusieurs stratégies sont possibles pour paralléliser l'extraction. Une stratégie simple est de constater que les descripteurs de fichiers sont partagés par l'ensemble des threads, et qu'il est donc possible de lire l'archive à extraire de n'importe quel thread. Chaque thread peut alors, alternativement, lire un fichier dans l'archive, avant de passer la main au thread suivant pendant qu'il écrit les données sur le disque dur.

Le nombre de threads utilisé peut être fixé (à 8 par exemple) dans le programme.

2.4 Interface utilisateur

Une fois compilé, votre projet se présentera sous la forme d'un fichier `ptar`, qui extraira dans le répertoire courant l'archive *tar* passée en paramètre. Exemple : `./ptar mon-fichier.tar`

2.5 Limites

Dans un premier temps, il n'est pas nécessaire de prendre en compte l'ensemble des types de fichiers pouvant être stockés dans une archive *tar*, ni l'ensemble des informations stockées pour chaque fichier (permissions, dates, ...). Toutefois, prendre en compte d'autres types de fichiers et des informations supplémentaires est une bonne manière d'étendre votre projet (voir ci-dessous).

Dans une première étape (avant extension), vous pouvez :
- vous limiter à l'extraction des répertoires et des fichiers "normaux" ;
- ignorer les permissions, les dates, etc.

2.6 Conseils de réalisation

La quantité de code à produire est relativement faible (quelques centaines de lignes de code tout au plus). Mais le niveau de difficulté du code à produire est très important. Il est crucial de programmer de manière prudente, réfléchie, claire, en testant bien les différents cas d'erreur.

Il est conseillé (mais pas obligatoire) de réaliser votre projet dans l'ordre qui suit.

Étape 1 : développement d'un *listeur* de fichiers tar.

Il est conseillé de commencer par un programme permettant simplement d'afficher le contenu d'une archive *tar*.

Pour cette première étape, il faut absolument bien lire la page de manuel de `tar(5)`, et éventuellement compléter cette lecture par d'autres documents. Il est conseillé d'utiliser le format *POSIX ustar archives* ou *GNU tar archives*.

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. Pour lire les en-têtes, il suffit de faire un `read()` dont la destination est une structure (`struct header_posix_ustar ma_struct; read(fd, &ma_struct, 512);`). Faites bien attention à la description des différents champs : certaines valeurs numériques sont en octal.

Étape 2 : développement d'un extracteur de fichiers *tar*.

Cette étape consiste à créer les répertoires et fichiers contenus dans l'archive à l'aide des différents appels système de manipulation de fichiers.

Étape 3 : ajout de la *durabilité*.

Cette étape consiste à s'assurer que les fichiers sont bien écrits sur le disque dur en ajoutant les appels à `fsync()`.

Étape 4 : parallélisation.

Étape 5 : extensions (cf ci-dessous).

3 Extensions possibles

Une réalisation absolument parfaite du projet décrit dans la partie précédente peut vous permettre d'obtenir une note entre 10 et 12. Pour aller au-delà, il vous faudra implémenter certaines des extensions suivantes.

Votre rapport précisera les extensions traitées par votre projet.

3.1 Gestion d'autres types de fichiers et d'autres informations

Difficulté : de très faible à importante selon les types de fichiers et les informations.

Ajoutez à votre code la gestion d'autres types de fichiers (liens symboliques, *devices*, liens durs, FIFOs) et des autres informations contenues dans l'en-tête TAR (utilisateur et groupe propriétaire, permissions (*mode*), date de modification, etc.). Dans votre rapport, indiquez quelles sont les informations supplémentaires que vous prenez en compte.

3.2 Gérer des options en ligne de commande

Difficulté : facile

Utilisez `getopt(3)` pour gérer quelques options, permettant par exemple :

- de choisir entre un mode verbeux (-v, verbose) ou un mode silencieux (-q, quiet) ;
- de changer de répertoire avant d'extraire l'archive (comme l'option -C de tar) ;
- de choisir le nombre de threads à utiliser ;
- d'activer ou désactiver certaines des extensions que vous avez ajoutées ;
- etc. (consultez `tar(1)` pour d'autres idées)

3.3 Gestion des archives compressées (gzip, bzip2, xz)

Difficulté : facile ou moyenne (avec *libdl*)

Après avoir détecté une archive compressée, utilisez l'API des bibliothèques de compression pour décompresser l'archive avant de l'extraire.

Une autre manière de procéder (plus intéressante, mais aussi plus difficile) est d'utiliser un système de greffon (avec *libdl*) pour charger dynamiquement la bibliothèque de compression nécessaire.

3.4 Vérification du champ *checksum*

Difficulté : assez difficile

Vérifiez que le contenu du fichier correspond bien au champ *checksum*.

3.5 Autres stratégies de parallélisation

Difficulté : assez difficile

Explorez d'autres stratégies de parallélisation, et comparez leurs performances respectives.

3.6 Meilleure durabilité

Difficulté : assez difficile

L'utilisation de `fsync()` lors de l'écriture des fichiers ne garantit pas que l'entrée du répertoire contenant le fichier a bien été écrite sur le disque. `fsync(2)` :

> *Calling fsync() does not necessarily ensure that the entry in the directory containing the file has also reached disk. For that an explicit fsync() on a file descriptor for the directory is also needed.*

Complétez votre programme.

3.7 Le reste...

Vous êtes libres d'ajouter toutes les idées d'extension que vous imaginez.

4 Calendrier

- Le mardi 01/11/2011 à 8h00, votre dépôt SVN doit contenir un fichier `AUTHORS`, un fichier `Makefile`, et les sources de votre projet. À cette date, votre projet doit (au moins) permettre de lister le contenu d'une archive *tar* ne contenant que des répertoires et des fichiers (correspondant à l'étape 1 ci-dessus).
- La version finale de votre projet est à rendre pour le **jeudi 01/12/2011 à 8h00**. Il sera récupéré directement sur vos dépôts SubVersion. Vous n'avez donc pas d'action particulière à effectuer pour *rendre* le projet, mais vous devez vous assurer que les fichiers requis sont bien présents. Une bonne manière de vérifier que tous les fichiers sont bien présents sur le dépôt est de réaliser un nouveau *checkout* et d'en vérifier le contenu. Les groupes dont le projet ne pourra pas être récupéré correctement seront évidemment sanctionnés.