LSINF1252 : Systèmes informatiques 1, Projet 3 : Extension de *cat* et réalisation d'un encodeur

de Mol Maxime, de Bodt Cyril, Groupe 19

16 mai 2013

Dans le cadre du dernier projet du cours de *Systèmes informatiques 1*, il nous a été proposé détendre la fonction *cat* dans la librairie *coreutils*, ainsi que d'implémenter l'algorithme *run-length encoding*. Le but du présent rapport est d'expliquer nos choix de conception ainsi que les problème rencontrés.

L'encodeur

Débutons par l'encodeur. Dans un souci d'efficacité, il a été demandé de traiter un fichier *input* via un mapping en mémoire. Cependant, dans l'hypothèse que l'on puisse recevoir des fichiers très gros en vue de les compresser, il s'est avéré que le mapping du fichier input devait se faire en plusieurs fois, pour ne pas saturer la RAM. Néanmoins, ceci a grandement complexifié la solution et ce fut l'un des problèmes auxquels nous furent confrontés : comment garder un code lisible et simple quand le nombre de cas et de possibilité à gérer devient très grand ? Nous avons tenté de trouver un compromis, sachant bien que la solution optimale n'existe peut-être pas.

Si nous tentons d'analyser le code de l'encodeur, celui-ci est structuré en cinq grosses parties. Premièrement, nous avons l'ensemble des variables globales à notre programme. Dans cet ensemble, nous avons d'abord des variables permettant de traiter les fichiers : les descripteurs de fichiers attachés à chacun de ceux-ci, une structure *stat* ayant des informations sur le fichier d'entrée, les pointeurs vers les zones correspondant au mapping des parties de fichier en mémoire, une variables définissant la taille maximale du fichier d'entrée pouvant être mappée en mémoire et enfin un pointeur vers la zone de la mémoire contenant un mapping de la fin de la zone précédante de l'output qui fut mappé en mémoire lors de l'itération précédente dans le fichier d'entrée.

Ensuite, nous avons un ensemble de variables gardant des informations sur la partie du fichier d'entrée traitée à l'itération précédente. Elles nous donnent par exemple le nombre de répétition du dernier octet traité à l'itération précédente dans le fichier d'entrée, quel était ce dernier octet, ... Enfin, le dernier ensemble de variables représentent des informations sur l'itération atuelle dans le fichier d'entrée : la taille du fichier d'entrée mappée en mémoire, l'offset courant dans le fichier d'entrée et de sortie (cet offset est un multiple de la taille des pages et ce n'est pas donc pas le « réel »offset), ...

La deuxième partie du code contient un ensemble de fonctions servant tantôt à ouvrir les fichiers, les fermer, comparer deux octets, ... Aussi, la fonction *use_give_nb_same_byte* permet de nous dire, à partir d'un double pointeur vers l'adresse de l'octet courant dans l'input, combien de byte identique le suive et d'incrémenter l'adresse du byte courant dans le fichier d'entrée de ce nombre.

La troisième gamme de fonctions dans l'encodeur est réservée au calcul de la taille de l'output nécessaire à la partie de l'input couramment mappée en mémoire. Nous trouvions plus élégant d'effectuer une itération de plus afin de connaître la taille nécessaire, plutôt que d'accorder trop de place et de tronquer l'output par la suite.

La quatrième partie du code nous permet de gérer la compression de l'input mappée couramment en mémoire. Durant cette section et la précédente, trois cas ajeurs se doivent être traités : soit l'octet courant de l'input est répéter moins que 128, soit il est répéter plus que 128 fois, soit il n'est pas répéter (ce qui correspond au cas où la variable *num* dans la fonction *compress* vaut 1). Aussi, une autre situation à gérer concerne le traitement de la fin de la partie mappée de l'output à l'itération précédente dans l'input. En effet, le dernier octet de la section prédemment mappée de l'input peut très bien être répété dans la nouvelle section du code alors qu'il ne l'était pas dans la section précédente. Ceci, ainsi que la multitude d'autres cas à traiter, complexifie grandement la solution. Pour nous aider, les fonctions *update_iter_prec* et *update_iter_prec_length_and_byte* nous aide à mettre à jour la fin de l'output mappé précédemment, en utilisant les variables globales gardant des informations sur le mapping précédant.

La dernière partie de l'encodeur contient les fonctions permettant de gérer les mapping successifs (*adjust_offset_output*) et leur traitement. Le lancement de l'encodage et du programme est assuré par les fonctions *make_encode* ainsi que par la *main*.

Pour résumer les difficultés que nous avons rencontrés dans l'élaboration de cet encodeur, nous pouvons dire que le fait de « multi-mapper »l'input a rendu la solution beaucoup plus complexe, même si cela s'avérait nécessaire en termes de performances. Aussi, signalons que nous avons également « multi-mappé »le fichier de sortie, qui est la version compressée. Pourquoi ? Eh bien, dans léventualité où le fichier d'entrée est très gros, il se peut que le fichier que nous produisons en sortie sature la RAM... Mais, une fois que le mapping par morceau du fichier d'entrée avait été réalisé, le faire pour le fichier de sortie était moins complexe.

Décodeur

Par rapport à l'encodeur, le décodeur fut beaucoup moins complexe. En effet, il nous suffisait de parcourir le fichier compressé et d'analyser les différents octets « length »pour savoir comment décompresser le fichier. L'analyse de ces octets « length »s'est faite selon le *run-length encoding*. Nous avons cependant dû faire face à un petit problème : le byte représentant un « length »de 10 est encodé de la même manière que l'octet qui fait un saut de ligne. Pour distinguer cet octet length »des vrais sauts de ligne, nous avons décidé d'encoder les octets de « length »10 comme des octets de « length »1, et ces derniers comme des octets de « length »129.

Extension de *cat*

Enfin, décrivons comment nous avons pu étendre la fonction *cat*. La difficulté de cette partie du travail fut de comprendre l'écriture de la fonction. Une fois ceci fait, il nous a alors suffit de comprendre où insérer les changements dont nous avions besoin dans la fonction. Alors, nous avons repris des conditions existantes dans *cat* pour les adapter à nos besoins, si jamais l'option « -x »était activée.