Licence d'informatique - Semestre 4 Automne 2015/2016
Système d'exploitation

Rapport de projet : Programmes d'archivage et de restauration

RABU Titouan
LUTTRINGER Jean-Romain

Sommaire :

I. Présentation du projet :

1. Fonctionnement et utilisation :

Le programme **backup** est appelé sur un répertoire racine d'une arborescence pour en créer une archive, dans un autre répertoire, composée d'un ou plusieurs fichiers selon les options à l'appel du programme.

backup [-d] [-i] [-z] [-p n] <repertoireRacine> <repertoireCible>

L'option **[-d]** permet, si elle est utilisée, de sauvegarder les dates d'accès et de modification des fichiers dans l'archive. Nous l'avons rendue obsolète en sauvegardant par défaut les dates.

L'option **[-i]** permet, si elle est utilisée, d'ignorer les fichiers spéciaux qui génèrent dans le cas contraire une erreur.

L'option **[-z]** permet, si elle est utilisée, de compresser la ou les archives créées au format *.gz.

L'option **[-p n]** générera un processus fils pour archiver chaque sous dossier du répertoire racine dans un autre fichier d'archive relié à l'archive principale. Le paramètre **n** de l'option définira un plafond de **n** processus fils que le père pourra exécuter simultanément.

Le paramètre **<repertoireRacine>** est le nom du répertoire que l'on souhaite archiver. Similairement, **<repertoireCible>** est le nom du répertoire que l'on souhaite créer pour contenir la ou les archives.

Le programme **restore** est appelé sur un répertoire contenant une ou des archives créées avec **backup** et restaure l'arborescence dans un nouveau répertoire passé en paramètre.

restore [-d] [-z] [-p n] <repertoireSource> <repertoireCible>

L'option **[-d]** permet, si elle est utilisée, de restaurer les dates d'accès et de modification des fichiers stockées dans la ou les archives. Sinon les fichiers et dossiers restaurés prendront comme dates l'heure courante.

L'option **[-z]** permet, si elle est utilisée, de restaurer uniquement les fichiers du **<repertoireSource>** qui sont compressés au format *.gz si il y en a.

L'option **[-p n]** générera un processus fils pour extraire chaque fichier d'archive différent de l'archive principale. Le paramètre **n** de l'option définira un plafond de **n** processus fils que le père pourra exécuter simultanément.

Le paramètre **<repertoireSource>** est le nom du répertoire où se trouve l'archive principale que l'on souhaite extraire. Similairement, **<repertoireCible>** est le nom du répertoire à créer dans lequel sera restaurée l'arborescence.

Dans notre **makefile**, la cible *check* exécutera uniquement ces deux tests :

```
./bin/backup -d -i -p 5 Dossier Archivage
./bin/restore -d -p 2 Archivage Restauration
```

2. Structure des fichiers d'archive :

Un fichier d'archive contient des entêtes stockant des informations sur lui-même et ses sous dossiers ainsi que ses fichiers, suivies de zéros binaires pour compléter sa taille jusqu'à 200 octets. Si un fichier n'est pas vide, son contenu est écrit à la suite de son entête et la fin de son contenu est complétée par des zéros binaires pour n'avoir que des blocs que 200 octets. La taille des blocs et des entêtes est définie par une variable globale qui permet une gestion bufferisée de la copie des données.

La structure des entêtes est commune aux dossiers et aux fichiers :

Nom du champ	***Taille dans l'entête***	***Description***
Nom	100 octets	Le chemin vers le fichier | dossier depuis le répertoire racine est stocké sur les 100 premiers octets, et est complété par des 0 binaires si il ne fait pas cette taille.
Droits d'accès	6 octets	Les droits d'accès sont stockés sous la forme *00xxxx* où *xxxx* correspond aux droits d'accès du fichier archivé en octal
Type	1 octet	Le type du fichier est stocké grâce à un entier. Ici, seulement les fichiers réguliers et répertoires sont traités, et respectivement sauvegardés par 1 et 5.
Taille	12 octets	La taille du fichier est stockée sous la forme d'un entier. Elle est utile pour la restauration du contenu des fichiers, et vaut 0 pour les dossiers.
Date de dernier accès au fichier	6 octets	La date de dernier est un entier qui sera converti en **`time_t`** lors de la restauration du fichier.
Date de dernière modification des données du fichier	6 octets	La date de dernier est un entier qui sera converti en **`time_t`** lors de la restauration du fichier.
Date de dernière modification des attributs du fichier	6 octets	La date de dernier est stockée comme un entier mais ne sera pas restaurée.

3. Structure des programmes :

a. Bibliothèques

Les programmes **backup** et **restore** utilisent tous deux les mêmes bibliothèques de fonctions en plus des fonctions qui leur sont propres.
Nous avons créé deux de ces bibliothèques en regroupant les différentes fonctions communes aux programmes **backup** et **restore**. Nous avons pris soin de n'utiliser que des primitives système pour la gestion des fichiers.

Fichier fonctions.c :

Contient les fonctions de gestion de fichier communes aux deux programme.

```
FICHIER * my_open(char * nom, char * mode) ;
int viderbuffer(FICHIER * f) ;
int my_getc(FICHIER * f) ;
int my_putc(FICHIER * f, int c) ;
int my_close(FICHIER * f) ;
int compresser(char * DossierSource) ;
```

Les fonctions **my_open**, **my_getc**, **my_putc** et **my_close** sont des fonctions analogues aux fonctions **open**, **getc**, **putc** et **close** que nous avons implémentées de telle sorte qu'elles fonctionnent avec un buffer.

Fichier chemin.c :

Contient des fonctions de gestions de chaînes de caractères et indirectement de déplacements dans une arborescence.

```
int taille(char * s) ;
int equals(char * s1, char * s2) ;
void remonterChemin(char * chemin, int slash) ;
void remonterCheminTotal(char * chem, int slash) ;
int nSubString(char * Str) ;
char* * split(char * Str) ;
void free2D(char* * Tab2D,int taille) ;
```

La fonction **taille** renvoie le nombre de caractère d'une chaîne.
La fonction **equals** vérifie que **s1** et **s2** sont identiques en taille et en contenu.
Les fonctions **remonterChemin** et **remonterCheminTotal** vont respectivement remonter au dossier parent et au dossier racine de l'arborescence passée en paramètre. Le paramètre **slash** est un flag qui permet ou non de garder le dernier caractère **'/'** dans le chemin.
La fonction **nSubString** compte le nombre de sous-chaînes dans **Str** délimitées par un caractère défini statiquement dans `chemin.h`.
La fonction **split** divise **Str** en un tableau de sous-chaînes selon le caractère défini statiquement dans `chemin.h`.
La fonction **free2D** désalloue un tableau de **taille** chaînes de caractères

b. *Programme backup :*

Le programme **backup** est le cœur de l'algorithme d'archivage. En plus des fonctions définies précédemment dans `fonctions.c` et `chemin.c`, d'autres sont spécifiques à son propre code source.

Fichier backup.c :

```
void   remplirBuffer(int   nBytes,char*   source,FICHIER*
destination) ;
void getType(int type, FICHIER* destination) ;
void ecrireEnteteFichier(FICHIER * f, FICHIER * Archive) ;
int TraiterDossier(char * dossier, FICHIER * Archive) ;
int TraiterFichier(char * fichier, FICHIER * Archive) ;
int wait_all(pid_t pid) ;
int parcoursRec(cha*Dossierource,charFichierCible[PATH_MAX],
int Fork, int nbProc) ;
int  SuperArchive(char*  DossierSource,  char*  FichierCible,
int FFork,int nbProc) ;
int  debut(char  *  DossierSource,  char  *  NomDossierCible,int
FFork,int nbProc) ;
```

La fonction **remplirBuffer** remplit le buffer de destination avec **nBytes** octets du buffer **source**.
La fonction **getType** appelle **my_putc** pour écrire le type de fichier type dans le contenu du fichier **Achive**.
La fonction **ecrireEnteteFichier** écrit l'entête du fichier `f` dans le fichier **Archive**.
La fonction **TraiterDossier** remplit une structure **FICHIER** et écrit son entête dans **Archive**.
La fonction **TraiterFichier** écrit l'entête du fichier dans **Archive** puis son contenu dans **Archive** .
La fonction **wait_all** attend la fin de toutes les processus fils avant de quitter.
Les fonctions **parcoursRec**, **SuperArchive** et **debut** sont les trois parties de l'algorithme récursif d'archivage.

Nous avons opté pour l'utilisation de **PATH_MAX** dans la déclaration de nos chemins pour éviter certaines désallocations de mémoire.

parcoursRec est la fonction principale de parcours récursif en profondeur de l'arborescence et elle archive toute cette dernière dans un fichier d'archive unique.

SuperArchive est une fonction non récursive qui sera appelée dans le cas où **backup** est exécuté avec l'option **[-p n]**. Elle va générer l'archive principale et créer des processus fils qui exécuteront **parcoursRec** sur les sous répertoires.

Chaque archive de dossier contient comme première entête les informations de ce dernier. Nous avons fait ce choix pour que le format d'archive coïncide avec notre méthode d'extraction.

Le programme **backup** se déroulera comme ceci :

- → Sans fork
  - → Création du dossier d'archivage dans **debut**
  - → Création de l'archive principale dans **debut**
  - → Lancement de **parcoursRec**
    - → Ouverture de l'archive et le dossier créés
    - → Appel de **TraiterDossier**
    - Tant que le répertoire n'est pas vide
      - → Lecture du répertoire
      - → Si c'est un fichier régulier
        - →Appel de **TraiterFichier**
      - → Si c'est un répertoire
        - →Appel de **TraiterDossier**
        - → Relancer **SuperArchive**
      - → Si ce n'est ni un répertoire ni un fichier régulier alors on génère une erreur si le programme a été lancé sans `[ -i ]` sinon l'erreur n'est pas générée
    - Fin Tant que
    - → Fermeture de l'archive
    - → Fermeture du dossier
  - → Sortie
- → Avec fork
  - → Création du dossier d'archivage dans **debut**
  - → Lancement de **SuperArchive**
    - → Ouverture de l'archive
    - → Appel de **TraiterDossier**
    - Tant que le répertoire n'est pas vide
      - → Lecture du répertoire
      - → Si c'est un fichier régulier
        - → Appel de **TraiterFichier**
      - → Si c'est un répertoire
        - → **NbrProcFils++**
        - → **FORK**
          - → **Dans le fils :**
            - → **Appel de** `parcoursRec`
            - → **Quitter le fils**
          - → Dans le père :
            - → Si **NbrProcFils** ≥ **MaxProcFils**
              - → **WAIT**
              - → **NbrProcFils--**
            - → Sinon rien
      - → Si ce n'est ni un répertoire ni un fichier régulier alors on génère une erreur si le programme a été lancé sans `[ -i ]` sinon l'erreur n'est pas générée
    - Fin Tant que
    - → Appel de **wait_all**
    - → Fermeture de l'archive
    - → Fermeture du dossier
  - → Sortie

c. *Programme restore :*

Le programme **restore** est le cœur de l'algorithme d'extraction d'archive. En plus des fonctions définies précédemment dans `fonctions.c` et `chemin.c`, d'autres sont spécifiques à son propre code source.

Fichier restore.c :

```
int wait_all(pid_t pid) ;
int estCompresse(char* f) ;
int decompresser(char* DossierSource);
FICHIER * lireEntete(FICHIER * Archive) ;
void creerFichier(FICHIER * f, char chemin[PATH_MAX]) ;
void creerRepertoire(FICHIER * dir, char chemin[PATH_MAX]) ;
int restore(char * Source, char chemin[PATH_MAX], int Ffork,
int nbProc) ;
int init(char * Source, char * Repertoire, int Ffork, int
nbProc) ;
```

La fonction **wait_all** est exactement la même que pour **backup** (cf. page 5). Cependant, nous n'avons pas réussi à la placer dans le fichier `fonctions.c` sans qu'elle ne présente des erreurs.
La fonction **lireEntete** lit un bloc de 200 octets dans une archive et stocke toutes les informations dans une structure **FICHIER** pour pouvoir le restaurer. Cette fonction n'est normalement appelée que quand l'offset du fichier archive se trouve au niveau d'une entête.
La fonction **creerFichier** crée un fichier à l'emplacement **chemin** avec les informations stockées dans **f**.
La fonction **restore** est une fonction qui s'occupe de l'extraction d'archives uniques ou multiples, et gérant également la création de processus fils pour l'extraction d'archives multiples dans le cas où l'option **[–p n]** est utilisée au lancement .
La fonction **init** crée le répertoire où restaurer l'arborescence et initialise **restore**.
restore fera appel à **estCompresse** et **decompresser** si jamais l'option **[–z]** est passée en paramètre.

Le programme **`restore`** se déroulera comme ceci :

→ Création du dossier d'extraction dans **`init`**
→ Si [-z] alors appel de **`decompresser`**
→ Appel de **`restore`**
→ Ouverture de l'archive
WHILE (1)
→ Appel de **`lectureEntete`**
→ Si son retour est NULL
→ alors **BREAK WHILE**
→ Si c'est un fichier régulier
→ Appel de **`creerFichier`**
→ Extraction du contenu
→ Si c'est un dossier
→ Si son archive existe
→ Si **`FlagFork`**
→ **`NbrProcFils++`**
→ **FORK**
→ **Dans le fils :**
→ **Appel de restore (sans fork)**
→ **Quitter le fils**
→ Dans le père :
→ Si NbrProcFils≥MaxProcFils
→ **WAIT**
→ **`NbrProcFils--`**
→ Sinon
→ Appel de **`restore`** (sans fork)
→ Sinon rien
END WHILE
→ Si **`FlagFork`**
→ Appel de **`wait_all`**
→ Fermeture de l'archive
→ Sortie
→ Si [-z] alors appel de **`compresser`**

II. Quelques modifications :

1. Archives numérotées :

Nous avons décidé de ne pas numéroter les archives secondaires qui sont créées en cas d'utilisation de **`backup`** avec des processus fils car nous avions commencé un algorithme qui gérait les archives secondaires par leur nom directement et pas avec un numéro de référence. De ce fait, nous ne stockons pas non plus une référence vers l'archive à traiter dans l'entête d'un dossier car **`restore`** détermine tout seul si cette dernière existe ou non.

Les plus grosses contraintes qui se sont présentées à cause de cette décision étaient qu'il fallait que l'archive principale créée par **`backup`** porte le même nom que le répertoire d'archivage, et qu'il fallait stocker les informations sur le répertoire qu'on archivait dans sa propre archive.

2. Sauvegarde des dates avec backup :

Dans notre version de **`backup`**, l'option **`[ -d ]`** n'a aucun effet. Nous nous sommes surtout concentrés sur cette même option lors du **`restore`**, ayant un impact direct sur la création des fichiers.

Ceci évite également des mauvaises gestions d'erreur quant à la récupération des dates si elles n'ont pas été sauvegardées dans l'archive, au détriment de l'optimisation de volume en cas de compression des archives avec l'option **`[ -z ]`**.

3. Magic Number :

Quand nous avons commencé notre projet, nous n'avons pas mis de Magic Number au début de nos archives, car nous n'en avions pas besoin tout de suite. Arrivés à la fin du projet, nous ne l'avions toujours pas dans nos archives mais nous n'en avions toujours pas besoin pour faire fonctionner nos programmes. Par manque d'utilité et de temps, nous n'avons pas jugé nécessaire de mettre le Magic Number au début de nos archives.

III. Problèmes rencontrés :

Durant toute la réalisation de notre projet, nous avons été confrontés à de nombreux problèmes.

- Nos archives lors de l'appel de la commande **backup** se créaient dans le répertoire courant à la place d'un répertoire portant le nom passé en paramètre de la commande. Quand nous avons voulu régler ceci pour correspondre au sujet nous avons dû effectuer un grand nombre de modifications sur notre code pour prendre en compte ce nouveau répertoire dans l'arborescence.
De plus, comme nous avions décidé de ne pas utiliser de numéro d'archive cela a engendré des problèmes supplémentaire (cf. page 10).

- Concernant la partie multi-procédurale des programmes, nous avons mal interprété le sujet, notamment à propos des options **[-p n]** et **[-P]**. En voulant coder la troisième étape nous nous sommes retrouvés avec un algorithme proche de celui demandé pour **[-P]** mais pas de celui demandé pour le sujet principal, et ce, la veille du rendu du projet.
Nous nous en sommes rendus compte en voulant gérer le nombre maximum de processus fils alors que notre algorithme générait des fils récursivement. Après une relecture attentive du sujet nous avons déduit que notre algorithme n'était pas le bon et nous avons donc dû reprendre la plupart de nos fonctions.

- Très vite, nous avions réussi à programmer un algorithme récursif (sans fork) pour **backup** qui générait une archive principale et des archives secondaires pour chaque sous répertoire. Cependant, il générait également des archives pour les sous-sous-répertoires et pas seulement pour les dossiers du répertoire racine, rajouté à cela des soucis d'implémentation du code pour fonctionner avec des fork.

- Une autre erreur d'inattention de notre part nous as fait nous rendre compte que nous n'avions pas pas compris la première demande du projet qui était de pouvoir générer une archive unique si **[-p n]** n'était pas utilisé alors que notre programme ne générait que des archives multiples. Nous avons donc dû modifier notre **restore** pour qu'il s'adapte également à cette demande.

- Une majeure partie de notre temps de travail fut consacrée au débuggage, mais surtout à la correction des fuites mémoires, qui étaient difficiles à détecter de part l'utilisation de processus fils.

- Un autre de nos problèmes a concerné la sauvegarde et la récupération des droits des fichiers. D'abord nous n'arrivions pas à mettre les bons droits dans l'archive avec **backup**, puis nous n'arrivions pas à récupérer les bons droits depuis l'archive avec **restore**, mais finalement, nous arrivons à sauvegarder tous les bons droits sans problème, et seul le droit d'écriture des utilisateurs (`---  ---  -w-`) n'est pas bien restauré avec **backup**, alors que la donnée est belle est bien récupérée dans l'entête de l'archive. Nous n'avons pas réussi à déterminer d'où venait l'erreur.