R. Namyst le 14 novembre 2016

Licence informatique — Programmation Système (4TIN503U)
Devoir Surveillé

Durée : 1h30 — Sans document

***Lisez l'intégralité du sujet** avant de commencer à répondre aux questions. Certaines peuvent être traitées même si vous n'avez pas répondu aux précédentes. À la fin du sujet, vous trouverez un memento vous rappelant la syntaxe de quelques appels système utiles.*

1 Questions de cours (échauffement)

Question 1 Expliquez ce qu'est un signal *masqué*. Que se passe-t-il lorsque plusieurs occurrences d'un signal sont envoyées vers un processus qui a masqué le signal ?

Question 2 Expliquez à quoi servent les fonctions `setjmp` et `longjmp` de la bibliothèque standard.

2 Manipulation de fichiers d'entiers

On souhaite écrire une suite d'utilitaires permettant de manipuler des fichiers d'entiers stockés au format brut (binaire). Ces fichiers débutent par un entier spécial « *magic number* » de valeur `0xcafebabe`. Dans toutes les questions suivantes, on utilisera l'interface de manipulation de fichiers offerte par les appels systèmes (i.e. `open`, `read`, `write`, etc.)

2.1 Calcul

On souhaite appliquer une fonction f à chaque élément d'un fichier d'entiers. En supposant qu'il existe une fonction prédéfinie `int f(int v)`, implémentez un programme `calcul` écrivant sur sa sortie standard les valeurs $f(x)$ correspondant à chaque x lu en entrée, après avoir vérifié que le premier entier lu est bien `0xcafebabe`. Les deux utilisations suivantes du programmes doivent être équivalentes :

```
calcul < input > output
```

```
calcul input > output
```

Si un nom de fichier est donné en argument, le programme redirigera son entrée standard depuis ce fichier dès le départ.

Dans l'exemple précédent, après exécution, les fichiers `input` et `output` ont la même taille et débutent tous deux par l'entier `0xcafebabe`.

2.2 Fichiers compressés

On suppose que les fichiers d'entiers manipulés contiennent de très nombreux 0. Aussi, on souhaite pouvoir les compresser un peu en proposant que toute suite de plusieurs 0 consécutifs soit remplacée par une seule occurence de 0 suivie du nombre de 0 qu'il faut ajouter pour recréer la séquence d'origine. Par exemple, si un fichier contient à l'origine la suite d'entiers suivante :

```
12 0 0 0 5 0 48 0 0 0 0
```

alors sa version compressée est :

```
12 0 2 5 0 0 48 0 3
```

Afin de distinguer les fichiers non compressés et les fichiers compressés, ces derniers possèderont le *magic number* `0xcafebaba`.
Attention : vous pouvez traiter au choix la question 2.2.1 ou la question 2.2.2. Dans les questions ultérieures, vous supposerez que vous disposez bien des deux implémentations.

2.2.1 Décompression

Donnez le code du programme `zcat-int` permettant d'envoyer sur la sortie standard la version décompressée d'une suite d'entiers :

```
zcat-int < input > output
```

Dans cet exemple, le fichier `output` débutera naturellement par l'entier `0xcafebabe`.
On s'attend à ce que la suite d'entier fournie en entrée soit compressée (c'est-à-dire qu'elle débute par `0xcafebaba`), mais si elle ne l'est pas, il faudra simplement que le `zcat-int` se contente de recopier l'entrée standard vers la sortie (dans l'exemple, les fichiers `input` et `output` seront donc identiques). On s'appuiera pour cela sur la commande Unix `cat`.

2.2.2 Compression

Donnez le code du programme `zip-int` permettant de compresser une suite d'entiers :

```
zip-int < input > output
```

Dans cet exemple, le fichier `output` débutera forcément par `0xcafebaba`.
De manière symétrique à `zcat-int`, si les données en entrée sont déjà sous forme compressée, `zip-int` recopiera simplement l'entrée vers la sortie en utilisant la commande Unix `cat`.

2.2.3 Calcul sur des fichiers compressés

On souhaite adapter le programme `calcul` défini à la question 2.1 de manière à ce qu'il fonctionne uniquement sur des fichiers compressés. Le fichier de sortie sera également compressé.
Profitez du fait que le fichier soit compressé pour diminuer le nombre d'appels à f : cette fonction étant supposée sans effet de bord, $f(0)$ retourne toujours la même valeur. Mais attention, vu que $f(0)$ ne retourne pas forcément la valeur 0, le fichier de sortie ne sera pas compressé « de la même façon » que l'entrée.
Donnez une nouvelle version du programme `calcul` en utilisant le programme `zip-int` pour produire le fichier de sortie.

2.3 Archives de fichiers

On souhaite pouvoir construire des archives de fichiers, c'est-à-dire des fichiers qui contiennent eux-même d'autres fichiers. Le format d'archivage est très simple et ne permet de mémoriser que des données (i.e. aucun nom de fichier n'est associé aux données). L'idée est la suivante : une archive est une concaténation de contenus de fichiers, chaque contenu étant précédé par un entier indiquant sa longueur (en octets). La figure 1 illustre la structure d'une archive dans laquelle on a d'abord ajouté un fichier de 128 octets, puis un second fichier de 52 octets.
La construction des archives s'effectue au moyen d'une commande `archive-add` qui permet d'ajouter des données à une archive, éventuellement après l'avoir créée. Cette commande prend le nom de l'archive en argument et lit les données à ajouter depuis l'entrée standard :

```
archive-add arch_name < input
```

Voici un exemple d'utilisation concret :

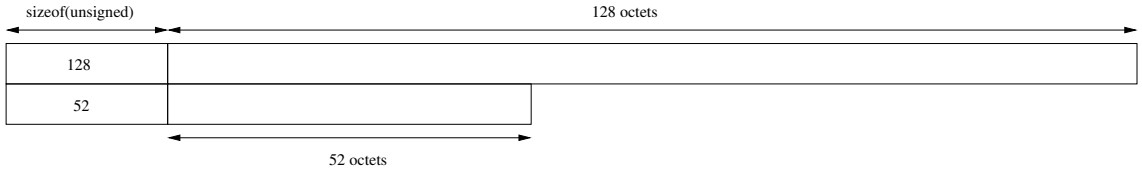

FIGURE 1 – Exemple d'archive contenant deux fichiers : un premier de 128 octets et un second de 52 octets.

```
[prompt] ls -l
-rw-r--r--  1 rnamyst  staff   128  9 nov 14:59 f1
-rw-r--r--  1 rnamyst  staff    52  9 nov 14:59 f2
[prompt] archive-add arch < f1
[prompt] ls -l
-rw-------  1 rnamyst  staff   132  9 nov 15:17 arch
-rw-r--r--  1 rnamyst  staff   128  9 nov 14:59 f1
-rw-r--r--  1 rnamyst  staff    52  9 nov 14:59 f2
[prompt] archive-add arch < f2
[prompt] ls -l
-rw-------  1 rnamyst  staff   188  9 nov 15:20 arch
-rw-r--r--  1 rnamyst  staff   128  9 nov 14:59 f1
-rw-r--r--  1 rnamyst  staff    52  9 nov 14:59 f2
```

2.3.1 archive-add

Donnez le code du programme `archive-add`. Veillez à écrire un code qui fonctionne quelque soit la taille des données en entrée.
Note : on rappelle qu'il n'est pas possible d'utiliser `lseek` sur l'entrée standard de manière générale...

2.3.2 archive-add-zint

En utilisant les commandes `archive-add` et `zip-int`, écrivez un programme `archive-add-zint` capable d'ajouter un fichier d'entiers à une archive en le compressant s'il n'est pas déjà compressé :

```
archive-add-zint arch < input
```

Memento

```
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
int open(const char *pathname, int flags, mode_t mode);
off_t lseek(int fd, off_t offset, int whence);
/* whence = SEEK_SET ou SEEK_CUR ou SEEK_END */
int dup2(int oldfd, int newfd);
int execlp(const char *file, const char *arg, ...);
int pipe(int fildes[2]);
pid_t fork(void);
```