Rapport du projet de système

DAZIBAO

Responsable de l'U.E. : Stefano Zacchiroli
Chargé de T.P. : Juliusz Chroboczek
Auteurs : Jérôme Tonnellier, Karim Ouharzoune et Mamadou Barry

1 – Organisation du projet

Le projet se compose de :

- D'un fichier ***README***.
- D'un fichier ***Makefile***.
- D'un fichier ***norme.h*** qui contient les constantes utiles au projet comme le champ *Magic* ou *Version* par exemple.
- D'un fichier source ***main.c*** qui permet d'afficher le menu principal et le sous-menu du projet. Il gère aussi la quasi totalité des fonctionnalités du projet.
- D'un fichier ***main.h*** y est associé pour plus d'explications.
- D'un fichier ***dazis-verifies.c*** qui permet de filtrer les dazibaos valides avant l'affichage du menu principal.
- D'un fichier ***dazis-verifies.h*** y est associé pour plus d'explications.
- D'un fichier ***verifie-entete.c*** qui permet de vérifier si un fichier est bien un dazibao.
- D'un fichier ***verifie-entete.h*** y est associé pour plus d'explications.
- D'un fichier ***modif-dazibao.c*** qui contient les fonctionnalités d'affichage, d'ajout, de suppression et de compaction d'un dazibao.
- D'un fichier ***modif-dazibao.h*** y est associé pour plus d'explications.
- Enfin d'un fichier ***service-notif.c*** qui contient le programme qui doit gérer les notifications de modifications des dazibaos.

2 – Extensions du projet

3 – Choix de conception

3.1 – Stockage de dazibaos valides (dazis-verifies.c)

Description :

Dans ce fichier, on y implémente une structure qui va simplement stocker les noms de fichier (char *) de dazibao(s) valide(s).

Voici donc cette structure :

```
struct dazis_verifies {
  int nb_dazis;
  char ** dazis;
};
```

La variable `dazis` contient ainsi la liste des noms des dazibaos corrects.

La variable `nb_dazis` contient donc le nombre de dazibaos. Elle permet de parcourir le champ `dazis` correctement.

But :

Cette structure est donc utilisée par le menu principal afin de filtrer les dazibaos non-corrects (elle est utilisée avant que le menu principal s'affiche).

3.2 – Fonctionnalités principales (`modif-dazibao.c`)

3.2.1 Fonctions d'affichage

- **La fonction d'affichage principale est :**

```
void affiche_dazibao(char * dazibao) ;
```

Description :

1 - Elle ouvre le `dazibao`.

2 - Elle tente de poser un verrou partagé sur celui-ci.

3 - Si tout s'est bien passé, elle déplace le descripteur de fichier juste après l'entête du fichier et récupère le nombre d'octets à lire.

4 - Elle appelle la fonction `affiche_tlv` qui va s'occuper du parcours récursif du dazibao.

5 – Elle enlève le verrou sur le fichier puis ferme le descripteur de fichier.

- **La fonction d'affichage récursive est :**

```
int affiche_tlv(int fd, int tailledonnees,
int num);
```

Description :

Pour chaque appel de fonction, `tailledonnes` est le nombre d'octets à lire par le biais du descripteur de fichier `fd`. Cette variable (`tailledonnes`) permet de savoir où s'arrêter dans le fichier. Elle est surtout utile pour parcourir les *TLV*s de type *Compound* et ainsi ne pas déborder sur les autres *TLV*s probables suivants.

La variable `num` permet de numéroter les *TLV*s. Ils possèdent alors un numéro de *TLV* suivant leur ordre d'apparition dans le `dazibao` (le 1er est 1, etc...).

3.2.2 Fonction d'ajout

La fonction d'ajout est :

```
int ajoute_tlv(
          char * dazibao, unsigned char typetlv,
             int length, char * fichierdonnees
);
```

Description :

Cette fonction permet d'ajouter un *TLV* à la fois.

1 - Elle ouvre le `dazibao` et positionne le descripteur de fichier à la fin de celui-ci.

2 – Elle tente de poser un verrou exclusif dessus.

3 – Si le verrou est posé, on ajoute un *TLV* à la fin du `dazibao` en fonction des cas suivant :

- 3.0 – Tout *TLV* :

Dans tous les cas, on écrit le type du *TLV* dans le `dazibao` en premier lieu.

- 3.1 – Cas du *Pad1* :

Pas grand chose à faire, on a écrit un 0 binaire en 3.0. On passe juste à l'étape 4.

- 3.2 – Cas du *PadN* :

C'est ici que l'argument `length` est utile, on transforme ce `int` en `char *` de 4 octets (en sachant qu'on ne va en utiliser que 3).
On écrit le champ *Length* dans le `dazibao` grâce au `char *` récupéré.
Puis on va écrire des données au hasard de taille `length` dans le `dazibao`.
On passe à l'étape 4.

- 3.3 – Cas des autres *TLV*s :

Même chose que le cas du *PadN*, sauf que pour copier les données,
on ouvre le fichier donné en argument `fichierdonnees` fourni par l'utilisateur.
On copie les données puis on passe à l'étape 4.

4 – Finalement, on enlève le verrou puis ferme le descripteur de fichier.

3.2.3 Fonction de suppression

- **La fonction de suppression principale est :**

```
int supprime_tlv(char * dazibao, int num);
```

Description :

1 – Elle ouvre le `dazibao`.

2 – Elle tente de poser un verrou exclusif dessus.

3 – Elle récupère la taille du `dazibao`.

4 - Elle appelle la fonction de suppression récursive `supprime_tlv_aux()`.

5 – Elle enlève le verrou et ferme le descripteur de fichier.

- **La fonction de suppression récursive est :**

```
int supprime_tlv_aux(int fd, int tailledonnees, int num);
```

Description :

On retrouve le même principe récursif que l'affichage. On compte les données grâce à `tailledonnees` pour ne pas dépasser ce que l'on doit réellement lire.

On compte le nombre de *TLV*s lus puis une fois qu'on tombe sur le *TLV* numéro `num`, on s'arrête puis on le supprime.

3.2.4 Fonction de compaction

La fonction de compaction est :

```
int compacte(char * dazibao);
```

Description :

La fonction va copier tous les *TLV*s autres que *Pad1* et *PadN* du fichier `dazibao` dans un fichier temporaire. Puis on va renommer ce dernier, au nom du `dazibao`.

1 – On ouvre le `dazibao`.

2 – On tente de poser un verrou exclusif dessus.

3 – Si le verrou est posé, on ouvre un fichier temporaire de nom douteux et on y copie l'entête.

4 – On va copier tous les *TLV*s du `dazibao` dans le fichier temporaire sauf les *Pad1* et *PadN*.

5 – On ferme le fichier temporaire

6 – On crée un second processus qui va renommer le fichier temporaire au nom du fichier `dazibao` grâce à la commande `execl` et `mv`.

7 – Enfin, on enlève le verrou puis on ferme le `dazibao`.

Justification :

On utilise un fichier temporaire pour la simplicité de programmation.
De plus, on utilise moins de mémoire que si l'on avait « mappé » la mémoire, donc cela ne pose pas de problèmes avec des fichiers de taille assez conséquente.