TP5

Gilles Menez - UNS - UFR Sciences - Dépt. Informatique

3 novembre 2018

---

1 Challenge Programming

Faire ces deux challenges :

① `https://51364960.widgets.sphere-engine.com/lp?hash=WkufwLACgf`

② `https://51364960.widgets.sphere-engine.com/lp?hash=zEaCpnaksl`

Dans les deux cas, les fichiers utilisés sont les fichiers standards : stdin et stdout.

2 id3v2.0

L'exercice qui suit est basé sur la structure des fichiers `mp3` et sur la technique des méta données permettant d'ajouter des informations à un contenu audio.

En réalité, ce qui m'intéresse

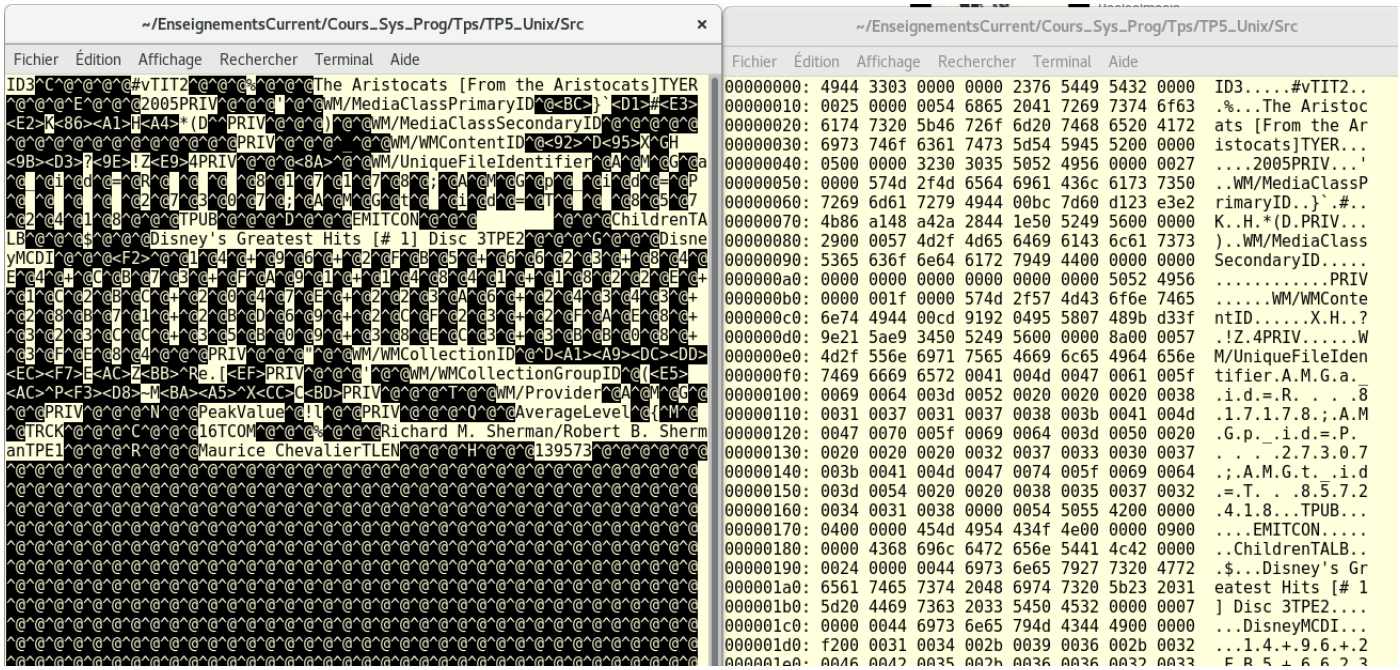
- ➢ c'est votre aptitude à analyser un fichier (le mp3 est anecdotique car toutes ces organisations sont "similaires"),
- ➢ puis à manipuler le système de fichier Unix.

Le "mp3" (MPEG Audio Layer III) est la spécification d'un format de compression audio avec perte :

- ➢ Compression ? . . . Logique que le fichier ne soit pas de type texte !

On peut "voir" un fichier mp3 :

`more track1.mp3` ou mieux `xxd track1.mp3`



2.1 Tag ID3v2.0

ID3v2 est un système de "tagging" qui permet de placer des "méta données" visant à enrichir le contenu musical d'un fichier audio.
Ces méta données sont contenues dans un `"tag"` décrit : `http://id3.org/id3v2.3.0`

D'un point de vue de sa position dans le fichier, le tag ID3v2 est un entête qui précède le contenu binaire des données audio.

Le tag ID3v2 est conçu comme un **conteneur de plusieurs blocs** d'informations : les `"frames"` .

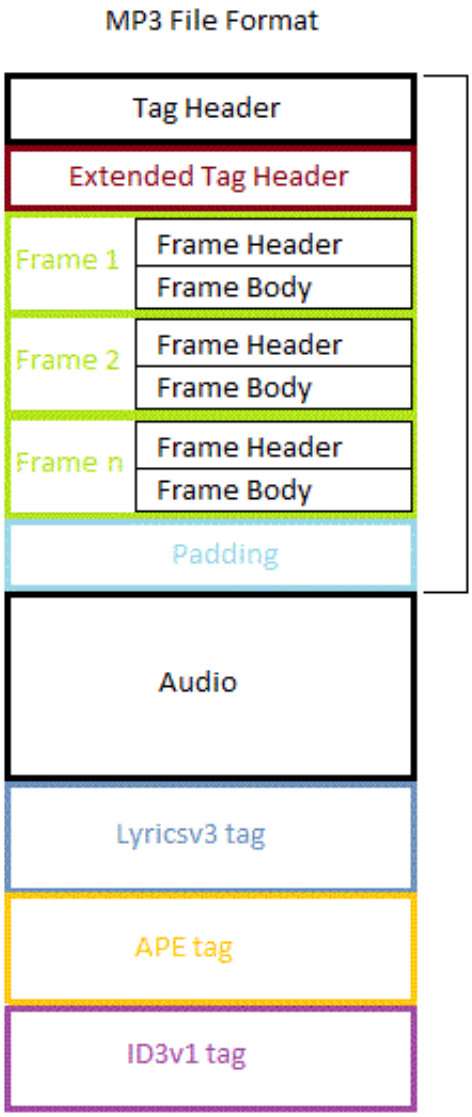
avec :

- ➢ en orange le "tag"
- ➢ en bleu , les différentes "frames", qui font partie du "tag".

Dans ces frames on peut trouver différents types d'informations :

- ✔ le titre,
- ✔ l'interprète,
- ✔ un site web,
- ✔ des images, . . .

Tous ces points donnent une structure au fichier mp3 qui ressemble (dans 99, 99% des cas de mp3) à ça :



Le "tag" a une structure : header + body (frames+padding)

Les "frames" contenues dans le tag ont elles aussi une structure.

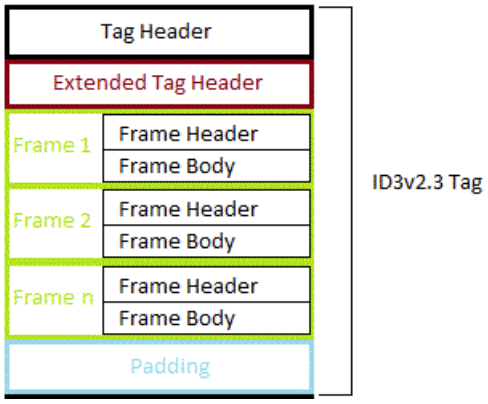
Au début de chaque frame, il y a un entête/header contenant :

- ✔ un identificateur,
- ✔ un descripteur de la taille qui permet au logiciel de lecture audio de sauter ce bloc,
- ✔ et un champ de flags qui décrivent les détails de l'encodage de la frame.

Cette figure vient de `http://beaglebuddy.com/` !

2.2 Structure du Tag

Le tag est donc le header du fichier mp3, MAIS ce tag a lui même une structure qui commence par un header !



2.2.1 Tag Header

L'entête du tag ID3V2 (/Tag Header), qui se trouve en début de tag occupe 10 octets (/bytes).

`http://id3.org/id3v2.3.0#ID3v2_header`

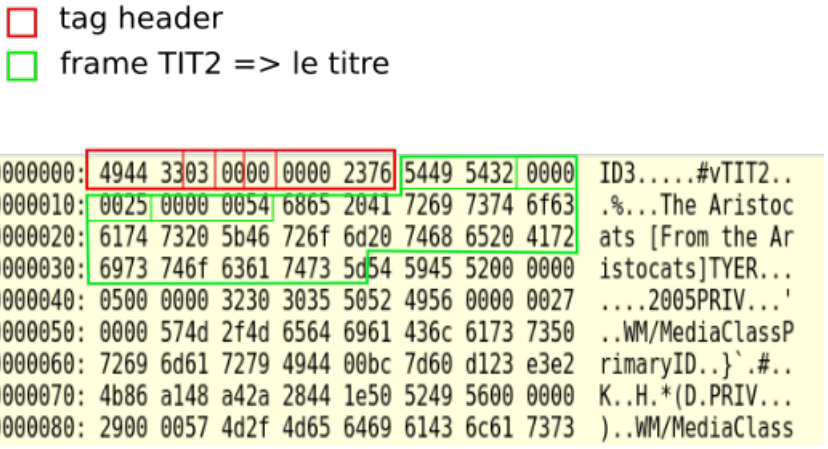
Ces octets représentent des informations de description du tag :

① Les 3 premiers octets sont les codes ASCII : "ID3"
C'est l'équivalent d'un "nombre magique" pour indiquer que l'on est bien face à un tag ID3v2.

② Les 2 octets suivants donnent la version du tag.
En premier octet, la "major" et en second octet la "revision" de cette major version.

③ Ensuite, 1 octet qui contient des "flags" (/indicateurs)

④ Enfin, 4 octets qui représentent la taille globale (en nombre d'octets) du tag id3 dans le fichier mp3 **moins les 10 octets du tag header**.



2.2.2 Taille dans le Tag Header

Il y a une petite particularité sur l'encodage utilisé pour représenter cette taille (un entier) :

➢ Le bit de poids fort de chaque octet (de ce groupe de 4) **n'est pas utilisé** et vaut 0.

Normalement, deux octets écrits en binaire utilisent 16 bits :

$$b_{15}b_{14}b_{13}b_{12}b_{11}b_{10}b_9b_8b_7b_6b_5b_4b_3b_2b_1b_0$$

qui contribuent ainsi selon la puissance associée à leur position :
$b_{15} * 2^{15} + b_{14} * 2^{14} + b_{13} * 2^{13} + b_{12} * 2^{12} + b_{11} * 2^{11} + b_{10} * 2^{10} + b_9 * 2^9 + b_8 * 2^8 + b_7 * 2^7 + b_6 * 2^6 + b_5 * 2^5 + b_4 * 2^4 + b_3 * 2^3 + b_2 * 2^2 + b_1 * 2^0 + b_0$

Ainsi par exemple = 0101010110101010 représentent le nombre :
$0*2^{15}+1*2^{14}+0*2^{13}+1*2^{12}+0*2^{11}+1*2^{10}+0*2^9+1*2^8+1*2^7+0*2^6+1*2^5+0*2^4+1*2^3+0*2^2+1*2^0+0$
$= 21930_{10}$

La spécificité du codage de la **taille au niveau du tag** est qu'il n'utilise que 7 bits d'un octet et le huitième est ignoré !

➢ Du coup, les puissances attribuées aux différents bits se décalent :

Ainsi par exemple, le bit 8 correspond à la puissance : 2^7

Toujours sur 16 bits, cela donne :

$b_{15} * 0 + b_{14} * 2^{13} + b_{13} * 2^{12} + b_{12} * 2^{11} + b_{11} * 2^{10} + b_{10} * 2^9 + b_9 * 2^8 + b_8 * 2^7 + b_7 * 0 + b_6 * 2^6 + b_5 * 2^5 + b_4 * 2^4 + b_3 * 2^3 + b_2 * 2^2 + b_1 * 2^0 + b_0$

Avec ce codage, une taille de tag = 00 00 02 01_{16} correspond à 257 octets (auquel il faut ajouter les octets du header (+10).

2.2.3 Padding

A la fin du tag, on voit qu'il peut (ce n'est pas obligatoire mais néanmoins fréquent) y avoir une zone de "padding" :

➢ Le padding, le "bourrage" en francais, permet de laisser un espace en réserve dans lequel de nouvelles frames pourrait être introduites sans avoir à re-écrire le fichier.

La taille du tag tient compte de ce padding.

Maintenant, pour l'analyse du fichier, la question est : "comment on sait qu'on entre dans le padding et que ce n'est plus une frame ?"

➢ Les octets du padding sont **par définition** "$0 \times 00_{16}$".

2.2.4 Extended header

On fait l'hypothèse qu'il n'y a pas de "extended header". Sinon ca complexifie un peu la chose ....

La présence d'un "extended header" est indiquée dans les flags du tag.

2.3 Frame

Aprés le header (du tag), il y a au moins une "frame".

`http://id3.org/id3v2.3.0#ID3v2_frame_overview`

2.3.1 Frame Header

Cette frame a, elle aussi, un header : 10 octets.

① 4 octets qui vont identifier la frame : des codes ASCII formants un **identificateur**.

Les codes ASCII autorisés (en usage courant) sont $['A' - 'Z'] \bigcup ['0' - '9']$.

`http://id3.org/id3v2.3.0#Declared_ID3v2_frames`

Certains identificateurs de frame sont "réservés" : Par exemple "TIT2", correspond à une frame qui va décrire :

"Title/songname/content description"

Les frames dont l'identificateur commence par la lettre "T" sont des `Text Information Frames` :

- ➢ Elles sont particulièrement informatives.
- ➢ Le texte contenu va décrire nom du compositeur ("TCOM") , nom de l'album ("TALB"), année de l'album ("TYER") ...

② 4 octets qui donnent la taille de la frame (moins les 10 octets du header de frame). Même codage que pour le tag header !

③ 2 octets qui correspondent à des flags/indicateurs.
Ces indicateurs indiquent par exemple si la frame est cryptée ou compressée ou ...

2.3.2 Frame Body pour les "Text Information Frames"

Ces frames vont donc être d'une grande aide pour analyser le contenu du fichier.

`http://id3.org/id3v2.3.0#Text_information_frames`

RMQ : il n'y peut y avoir qu'une frame de son espèce dans le fichier ! (donc par exemple au plus une frame "TIT2" par fichier)

Ces frames ("Text Information") ont la même structure :

① Aprés l'IDentificateur de la forme "TXYZ" (soit 4 octets),

② 1 octet pour donner le type de codage utilisé pour le texte qui va suivre,

③ Les octets jusqu'à la fin de la frame (vous vous souvenez que l'on connait sa taille !).

2.4 Votre travail

Analyser et comprendre le code qui vous est donné !

Dans le fichier `main.c` , il y deux thématiques. La première partie consiste à être capable de parcourir le contenu d'un fichier mp3. C'est le rôle de la fonction `mp3_read` (dans main.c) : il faut la comprendre pour faire la suite. Donc On pose des questions !!!!!

Dans cette fonction,

➢ On commence par y lire le header du tag : ce qui permet de connaitre la taille du tag.

Attention au codage de cette taille !

➢ Ensuite, on lit les frames une par une dans une itération.

La valeur de retour de ces fonctions de lecture permet de progresser ou de s'arrêter dans le fichier. On s'inspire/adapte de la fonction `read(2)` :

- ✔ -1 : erreur pendant la lecture
- ✔ 0 : plus rien a lire
- ✔ sinon la taille de ce que l'on a lu.

Attention à l'arrêt sur le padding !

① En soit, ce programme ne sert à rien puisqu'il ne fait que "traverser" le fichier mp3.

Vous allez le transformer pour coder une fonction qui rend le contenu d'une frame de type "text information" dont on vous donne l'idenfificateur.

Je vous aide un peu en vous donnant le prototype et le mode d'emploi :

```
char *id = "TIT2";
char contenu[10000] = ""; /* On fait simple sinon il faudrait
un malloc quelque part dans les fonctions appelées */
int cr = mp3_get_frame_from_id(fd, id, contenu);

if ((cr != -1) && (contenu[0] != '\0')){
  contenu[30] = '\0';
  printf("%s \n",contenu);
}
else{
  printf("%s \n", "pas trouvé");
}
```

Ce n'est pas une version optimale puisqu'on doit faire une hypothèse sur la taille maximale de la frame . . . faisons déjà comme cela !

Au retour de l'appel :

- ✔ si cr vaut -1, c'est qu'il a eu un problème (système ?), sinon il sera différent de -1
- ✔ si "contenu" est encore une chaine vide, c'est que la recherche de cet identificateur de frame n'a rien donné dans le fichier.

2.5 MP3 organizer

Maintenant que vous savez accéder aux informations du tag, on peut aborder les choses sérieuses et accéder à la deuxième thématique du TP : la manipulation du SGF.

Un peu comme pour : `https://help.mp3tag.de/main_converter.html`

Imaginons que vous ayez rippé/encodé votre album favori :

➢ On simulera votre "album favori" avec un album issu de `http://freemusicarchive.org/` ... histoire de ne pas avoir de problème de droit commercial.

Mais malheureusement, les fichiers obtenus ont des noms en "track1.mp3", "track2.mp3, ... contenu dans un répertoire "MP3".

➢ Bref, pas super le classement !

Même si le nom du fichier n'est pas explicite, il y a de grandes chances que les méta données soient correctes. Trois types de frames vont nous intéresser :

➢ "TIT2"

➢ "TALB"

➢ "TRCK"

pour créer un répertoire dédié à l'album et des noms de fichiers plus explicites.

Pas à pas :

① On fait l'hypothèse que dans le répertoire MP3, il n'y a que les fichiers mp3 qui nous intéressent (c'est à dire du même album). Par contre, il peut avoir d'autres types de fichiers.

En utilisant le petit exemple fourni dans le cours sur "les E/S sur répertoire", êtes vous capables d'afficher uniquement les noms des fichiers en ".mp3" ?

Ce qui lien peut être utile ?

`https://stackoverflow.com/questions/4849986/how-can-i-check-the-file-extensions-in-c`

② Ok, vous savez les afficher, mais est ce que vous pouvez renvoyer un tableau de chaînes de caractères contenant ces noms ?

```
char **lf = ls_mp3files_inarray("./MP3"); /* renvoie les .mp3 dans
                                             le répertoire ./MP3 */
i = 0;
while(lf[i] != NULL){
  printf("%s\n",lf[i]);
  i++;
}
```

Petit indice : pour faire le malloc du tableau dans la fonction `ls_mp3files_inarray()` , j'ai parcouru le répertoire pour compter les fichiers mp3, puis j'ai alloué le tableau et enfin j'ai re-parcouru le répertoire pour remplir le tableau.

③ Puisqu'on fait l'hypothèse simplificatrice que tous les fichiers mp3 présents dans le répertoire courant appartiennent au même album.

Faire une fonction qui récupère ce "nom d'album" à partir de n'importe quel fichier mp3 de ce répertoire ... le premier ?

Si vous n'y arrivez pas ... faites au moins une fonction qui demande le nom au clavier et qui l'utilise pour la question suivante.

④ Créer un répertoire de ce nom (ou des 10 premieres lettres de ce nom) dans le répertoire "MP3".

⑤ Déplacer les fichiers mp3 de cet album dans le répertoire que vous venez de créer.

Plusieurs solutions :

➢ `https://stackoverflow.com/questions/22856040/how-to-move-the-files-from-one-directory-to-anothe`

➢ ou alors par la fonction system().

⑥ Lorsque vous savez faire, ce déplacement pourrait aussi être l'occasion de modifier le nom des fichiers en utilisant les informations du tag pour les renommer :

"TIT2_TRCK".mp3

A la fin de l'exécution de votre programme, les fichiers sont regroupés dans un nouveau (si il n'existait pas déja) répertoire portant le nom de l'album.

Dans ce répertoire, les fichiers ont des noms qui utilisent les informations issues des frames TIT2 et TRCK :