CS221 - TP2

Vincent MOUCADEAU - Rémi MAZZONE — 2A

30/11/2022

Table des matières

Listings

1 Introduction

Dans ce TP, nous allons travailler sur les fichiers et les applications qu'on peut y trouver dans le domaine de la stéganographie.

2 Travail Préparatoire

2.1 Nombres impairs

Listing 2.1: Code pour avoir les 100 premiers nombres impairs

```c
void file_odd(int n) {
    int i = 0, c = 0;
    FILE *f = fopen("nombres_impairs.txt",
       "w");
    while (c < n) {
        if (i % 2 != 0) {
            fprintf(f, "%d\n", i);
            c++;
        }
        i++;
    }
    fclose(f);
}
```

2.2 Comptage de voyelles

Listing 2.2: Code pour compter le nombre de voyelles dans un fichier texte

```c
int file_count_vowels(char *filename) {
    int count = 0;
    char vowels[6] = {'a', 'e', 'i', 'o', '
       u', 'y'};
    int tab_len = sizeof(vowels)/sizeof(
       vowels[0]);
    FILE *file = fopen(filename, "r");
    char curr_char;

    while (curr_char != EOF) {
        curr_char = fgetc(file);
        for(int i = 0; i < tab_len; i++) {
            if(curr_char == vowels[i]) {
                count++;
                i = tab_len;
            }
        }
    }
    fclose(file);
```

```
18
19      return count;
20  }
```

2.3 Fichier transporteur into source

Listing 2.3: Algorithme en français de ce processus

```
1  input : fichier f_input
2  output : fichier f_output
3  ouvrir f_input
4  octet = 0
5  nouveau_char = ""
6  tant que f_input pas termine
7      si octet = 8
8          nouveau_char <- conversion bin to char de
               nouveau_char
9          ecrire nouveau_char dans f_output
10     sinon
11         evaluer le prochain caractere de f_input
12         si il est alphanumerique
13             si c'est une majuscule
14                 nouveau_char <- nouveau_char + '1'
15             sinon
16                 nouveau_char <- nouveau_char + '0'
17             fin si
18             octet++
19         fin
20     fin si
21 fin tant que
```

2.4 Makefile

Listing 2.4: Makefile du projet

```
1  main: main.o
2          gcc -o $@ $^ -g
3
4  main.o: main.c
5          gcc -c $< -g
6
7
8  clean:
9          rm -rf *.o main
```

3 Stéganographie

3.1 Une application un peu trop visible...

À partir du code rédigé en pseudo code dans la partie préparatoire (cf. 2.3), nous avons pu écrire le code correspondant en C. Le voici :

Listing 3.1: Code décodage fichier txt

```
void readable_txt(char *filename) {
FILE *f = fopen(filename, "r");

char *main = malloc(strlen(filename) + strlen("_source.txt") + 1);
char *second = malloc(strlen("_source.txt") + 1);
strcpy(main, filename);
strcpy(second, "_source.txt");
FILE *fout = fopen(strcat(main, second), "w");

char new_char[8];
int octet = 0;
int curr_char = 0;
while (curr_char != EOF) {
    if(octet == 8) {
        fputc(bit_to_int(new_char), fout);
        octet = 0;
    } else {
        curr_char = fgetc(f);
        if(isalpha(curr_char) != 0) {
            if(isupper(curr_char)) {
                new_char[octet] = 1;
            } else {
                new_char[octet] = 0;
            }
            octet++;
        }
    }
}
fclose(f);
fclose(fout);
}
```

Code code va donc parcourir le fichier transporteur et va écrire dans un nouveau fichier le message caché. Pour cela, il va lire le fichier transporteur caractère par caractère. Si le caractère lu est une lettre, il va regarder si c'est une majuscule ou une minuscule. Si c'est une majuscule, il va écrire un 1 dans un tableau, sinon il va écrire un 0. Il va continuer à faire cela jusqu'à ce qu'il ait lu 8 caractères (ce qui correspond donc à 1 octet car on aura inséré 8 bits dans le tableau). Ensuite, il va convertir la chaîne de caractères en un entier et l'écrire dans le fichier de sortie. Il va continuer à faire cela jusqu'à la fin du fichier principal et on aura donc fini de décoder le fichier !

3.2 Une application un peu moins visible...

On a donc 3 champs de structures dans le fichier bitmap.h : fichierEntete, imageEntente et couleurPalette. Pour les lire et les afficher, on utilise le code suivant (l'exemple ne montre la procédure que pour un champ avec l'affiche de seulement quelques informations mais c'est la même chose pour les autres) :

Listing 3.2: Code pour lire un champ de la structure bitmap

```
FILE *f = fopen(filename, "r");
fichierEntete *header = malloc(sizeof(fichierEntete));
fread(header, 1, sizeof(fichierEntete), f);
printf("%d\n", header->tailleFichier);
printf("%d\n", header->offset);
fclose(f);
```

En partant de ces informations, on a donc pu imaginer le code qui nous sera nécessaire à décoder l'image. On va donc parcourir tous les pixels de l'image source (en commencant après tous les entêtes d'un fichier Bitmap) et on va en extraire le bit de poids faible à chaque fois. Ainsi, on va pouvoir former petit à petit l'image qui avait été cachée.

Listing 3.3: Code pour décoder une image

```
char *decode_bmp(char *filename) {
FILE *f = fopen(filename, "r");

char *main = malloc(strlen(filename) + strlen("_source.jpg") + 1);
char *second = malloc(strlen("_source.jpg") + 1);
strcpy(main, filename);
strcpy(second, "_source.jpg");
FILE *fout = fopen(strcat(main, second), "w");

fichierEntete *header = malloc(sizeof(fichierEntete));
imageEntete *imageHeader = malloc(sizeof(imageEntete));

fread(header, 1, sizeof(fichierEntete), f);
fread(imageHeader, 1, sizeof(imageEntete), f);
fseek(f, header->offset, SEEK_SET);

char pixel, bit[8];
int octet = 0;
for(int i = 0; i < imageHeader->tailleImage; i++) {
    pixel = fgetc(f);
    bit[octet] = get_bit_faible(pixel);
    octet++;
    if(octet == 8) {
        fputc(bit_to_int(bit), fout);
        octet = 0;
    }
}

fclose(f);
```

```
30  fclose(fout);
31
32  printf("File decoded succefully\n");
33
34  return main;
35  }
```

Ce code utilise quelques fonctions annexes que nous avons codés afin de rendre cette partie plus compréhensible. Voici donc les extraits des fonctions nécessaires :

Listing 3.4: Fonctions annexes

```
1   char get_bit_faible(char octet) {
2       char mask = 1;
3       return mask & octet;
4   }
5
6   int puissance(int a, int b) {
7       int res = 1;
8       for(int i = 0; i < b; i++) {
9           res *= a;
10      }
11      return res;
12  }
13  int bit_to_int(char bit[8]) {
14      int tot = 0;
15      for(int i = 0; i < 8; i++) {
16          if(bit[i] == 1) {
17              tot += puissance(2, 7-i);
18          }
19      }
20      return tot;
21  }
```

À la suite de l'éxécution de cette fonction, nous avons donc obtenu l'image suivante :

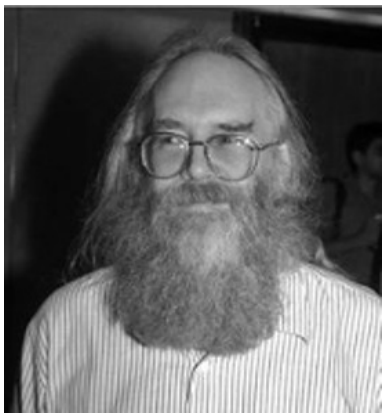

Figure 1: L'image cachée

En faisant une rapide recherche d'image inversée sur Google, on a découvert qu'il s'agit en fait de Jonathan Bruce Postel, dit Jon Postel, un célèbre informaticien qui a contribué à la création d'Internet.
Maintenant que nous avons décodé l'image dans la question précédente, nous allons faire la procédure inverse : nous allons recréer l'image que nous venons de décoder. Dans la question précédente, nous avons pris l'image transporteur.bmp et nous en avons décoder l'image jon_postel.jpg. Désormais, nous allons nous servir de l'image originel.bmp dans laquelle nous allons encoder l'image jon_postel.jpg pour reconstituer le fichier transporteur.bmp.

4 Conclusion

Lors de ce TP,