TINAZZI Giovanni VANSON Nathan

RAPPORT TP2

Exercice 2.1

La commande “sha1sum” est une fonction de hachage ainsi que “md5sum”. Pour rappel, utiliser la commande “man” nous permet d'accéder au manuel de ce que l’on demande (ex: “man md5sum” nous permet d'accéder au manuel de “md5sum”)

En utilisant les commandes “sha1sum” et “md5sum”, on peut calculer les hash du contenu du fichier. Ainsi, les résultats sont:

pour md5sum: 156b99a548d3ad54e3eb31bb89f73a5e text_Exo2.txt

pour sha1sum: 5d57bae00b65a3e3cdfc59501cedaadf7191f456 text_Exo2.txt

Sans utiliser de fichier et en combinant les commandes, le résultat est différent. Ici, le problème vient d’un saut de ligne. En effet, un éditeur de texte comme nano ne met pas forcément de saut de ligne, ce qui change le calcul de hachage. Pour résoudre le problème, il faut utiliser l’option “-n” de la commande echo qui va nous permettre d'éviter le saut de ligne d’echo et donc d’avoir le même texte exactement. Notre commande devra ressembler à ca:

```
    echo “Le manuel disait: Nécessite Windows 7 ou mieux. J’ai donc
installé Linux” | sha1sum
```

Exercice 2.2

En utilisant la commande “man 3 EVP_DigestInit” on va pouvoir accéder au manuel de EVP_DigestInit, dans la section 3, c'est-à- dire dans les librairies C. Dans ce manuel, un exemple nous est fourni permettant de calculer le digest/hash de fichier et de chaînes de caractère. Pour implémenter le programme, il suffit de ouvrir un fichier C dans l’editeur que l’on veux;

Puis pour compiler le programme en liant les librairies libssl et libcrypto, on va taper dans le terminal la ligne suivante:

```
    gcc -o Exo_2.2 Exo_2.2.c -lssl -lcrypto
```

Ici, gcc est un compilateur de C et C++. L’option “-o” permet de renommer notre fichier .c et un fichier compile qu’on pourra utiliser par la suite. par la suite en utilisant “-l” on va pouvoir lier les librairies, qui sont indispensables pour le fonctionnement du programme.

Enfin, en testant le programme avec le texte: “Le manuel disait: Nécessite Windows 7 ou mieux. J'ai donc installé Linux.”, et en utilisant comme algorithme de digest md5 par exemple, notre programme va afficher

en output:

```
Digest is: 6752701ccb975ab428e4a30d391971ba
```

Exercice 2.3

Dans cette partie du TP, on s'intéresse à la gestion de paramètres d'un programme. Tout comme l'exo précédent, on va pouvoir acceder au manuel de getopt dans la section librairie C avec la commande:

```
man 3 getopt
```

Une fois dans le manuel, on a une description de la fonction (getopt) et un exemple d'utilisation en C de cette fonction. Les point cles de ce programme sont:

```
Ici getopt est appelle en boucle. Quand getopt retourne -1, cel a signifie
qu'il n y a plius d'options de commande à tester. Une fois -1 retourne, la
boucle s'arrete.

Le swtich ici va etre utiliser pour trier les valeur retourne par getopt.
Dans un programme avec un but precis, chaque "case" va representrer une
variable du programme.

La deuxieme boucle est utilise pour traiter les argument restants qui ne
sont pas des options
```

Fonctionnement du programme

Apres avoir fait un `make all` pour pouvoir compiler et lier nos fichiers .c, on va avoir plusieurs options. Tout depend si on veux generer un hash de fichier ou d'une chaine de caractere. Lorsque l'on veux travailler sur un ou plusieurs fichiers, on pourra utiliser le programme de la maniere suivante:

```
./digest -f fichier1 fichier2....
```

De ce fait, l'option -f va permettre a notre programme de savoir que "fichier1" n'est pas une chaine de caractere, mais bien un fichier qu'il devra ouvrir pour generer le hash de la chaine de caractere presente dans ce fichier.

Si maintenant on s'interesse à une chaine de caractere, la commande a taper dans le terminal est la suivante:

```
./digest blabla blabla
```

Sans l'option -f, le programme va comprendre qu'il s'agit d'une chaine de caratere donc va generer un hash pour tout ce qui suit "./digest" Ici, pour que notre hash soit juste, il faut penser a faire un

```
EVP_DigestUpdate(mdctx, argv[optind], strlen(argv[optind]));
```

a chaque fois qu'on est sur une nouvelle chaine de caractere. Pour bien gerer l'espace à la fin pris en compte par le saut de ligne, on va tester si y a un espace en fin de ligne ou pas:

```
if(optind != argc -1){
            EVP_DigestUpdate(mdctx, " ", strlen(" "));
}
```