# Systèmes d'Exploitation - Examen

## 12X009 - TP02
## hashing & OpenSSL

Noah Munz (19-815-489)

Département d'Informatique
Université de Genève

Mardi 31 Janvier 2023

# Table of Contents



# Table of Contents



# Rappel: But du TP

- Se familiariser avec la manipulation de chaînes de caractères via la manipulation de `argv` / `argc` ainsi qu'avec le parsing d'option & paramètre ( `getopt` , `optstring` ...)
- Se familiariser avec la liaison de libairies externes (openssl), Makefile...
- Se familiariser avec l'utlisation fonctions de hashages

# Table of Contents



# TADs & leurs relations: Décomposition modulaire

TP reste assez simple, seulement 2 modules ont été créés.
Le premier `OptionParser` (qui, part la suite, sera le début d'un module `util` qui sera réutilisé et aggrandi à chaque TP suivant) qui s'occupe de vérifier si l'input de l'utilisateur est valide ou pas, *sépare*, *copie* et *stoque* les différentes parties des différentes entrées en fonctions des options de ces dernières.

Le second `hash_calc` qui s'occupe de gérer les "digest context", i.e. les créer, y rajouter du text à hash... ces contexts sont une interface qui va permettre d'ajouter différents message à la suite puis d'en hash le tout uniqument lorsqu'on décide de le terminer.

# TADs & leurs relations: Décomposition modulaire

Par exemple `OptionParser` contient une fonction `checkEnoughArgs()` (dont le nom est déjà assez explicite), ainsi qu'une autre fonction `int parseArgs(int argc, char* argv[], char** fileToHash[], int* fileAmnt, char** stringToHash )` qui va :

- parse les arguments optionnels
- Si `-f` n'as *pas* été fourni $\Rightarrow$ appel une fonction plus simple pour juste hash la concaténation des entrées avec la méthode donnée avec `-t`. i.e. va stocker la concaténation dans `stringToHash`.
- Si `-f` a été fourni, va extraire chaque nom de fichier et les stocker dans le buffer `fileToHash`.

# TADs & leurs relations: Décomposition modulaire

`hash_calc`, quant à lui, contient une fonction `convert_f_to_s()` qui extrait le contenu d'un fichier pour pouvoir passer le contenu directement à la fonction `hash()`. L'implémentation de `convert_f_to_s()` a été repris du tp sur ultra-cp qui est elle-même fortement inspirée des slides du cours. i.e. implémentation manuelle "bufferisé" de lecture de fichier avec `read()`.

# TADs & leurs relations: Structures de données utilisées

L'implémentation de structure n'a pas été nécessaire.

# Table of Contents



# Tests realisés pour valider le fonctionnement du TP

Les tests réalisés ont été les suivants:

1. `./digest -f file1 file2 ...[-t <hashMethod>]`
   - `./digest -f res/test.txt res/string-tohash.txt -t md5`
   - output:

     `Hashing Method:  md5`

     `Hashing file "res/string-tohash.txt"...`

     `120227d6118cfddbd21639644aa0884d File:  res/string-tohash.t`

     (ou `ef81ae80ec507096f761cefa49f7b63e` sans retour à la ligne final)

     `------------`

     `Hashing file "res/test.txt"...`

     `4e1243bd22c66e76c2ba9eddc1f91394e57f9f83 File:  res/test.tx`

     `------------`
2. `./digest string1 string2 ...[-t <hashMethod>]`.

# Table of Contents



# Réponses aux questions (de l'énoncé du TP)

- Combiner la commande echo "Le manuel disait: Nécessite Windows 7 ou mieux. J'ai donc installé Linux". Avec les commandes ci-dessus pour calculer les hashs sans utiliser de fichier; le résultat est différent, pourquoi? Comment résoudre le probléme?
  - On voit que lcs 2 hashs different dans les 2 cas car on a un retour la ligne dans le fichier mais pas lorsque l'on pipe directement l'output de echo dans l'input dc SHA1 et MD5. En effet, on a vu avant qu'en ajoutant ou enlevant un saut de ligne à la fin du fichier `string-tohash.txt` on pouvant switcher comme on voulait entre ces 2 variantes possibles.

# Table of Contents



# Réponses aux questions (générales)