Collected logs et comptes rendu

21/02: Demande du premier rdv Client:
Pour identifier le besoin principal du projet et obtenir quelques clarifications supplémentaires de la part du client

28/02: Pas de rendez-vous, oubli de confirmation communication mail:

1. le test devra executer le fichier executable. Il s'agit vraiment de tests de comportement et pas de test unitaires / d'intégrations dans la mesure ou je laisse l'architecture du code libre pour les étudiants. En d'autre terme il s'agit de vérifier que le programme se comporte comme attendu lors de l'exécution.
Pour le client, il faudrait:
- tester les hash de chaînes;
- tester le hash des fichiers.
Cela devra être fais avec les hash SHA1 et MD5. La sortie du programme devra être parsée pour récupérer le résultat et le comparer au résultat des commandes sha1sum et md5sum.

A partir d'un code qui fonctionne, le client propose la stratégie suivante:
- développer les tests pour que le programme C passe les tests;
- changer le programme C pour générer des erreurs et vérifier que les tests sont bien "failed";
- m'envoyer le fichier de test pour que je l'essaye sur mon propre code.

06/03:
J'ai choisi d'utiliser les librairies:
1. unittest pour créer une classeTestCase, et pour l'assertion
2. subprocess pour l'interaction avec le shell
3. hashlib pour faire les hashs en Python, et comparer le résultat avec l'output

En ce moment, je fais le test_file_hash() avec des fichiers existant, mais je suis entrain de lire la documentation de tempfile pour que puisse créer plusieurs fichiers temporaires et tester mon prog avec.

Les soucis en ce moment :
1. Comment appeler correctement les tests. (j'ai initialisé une classe t=Test(), et essayé d'appeler mes fonctions, ça ne donne rien)
2. Apprendre à faire le wrapper
3. Apprendre à gérer les erreurs
4. Faire le test_md5()

07/03: Contraintes mises par le client:

« il faudrait que votre solution emplois le moins possible de librairies non standard (i.e. non installée avec le package python de base). En effet je souhaiterais que les étudiants puissent les utiliser sans "se prendre la tête" avec l'installation. Vous noterez que j'ai utiliser deux librairies non standard dans les scripts que je vous ai donnée: "colorama" et "psutil". Si vous trouvez un moyen de remplacer colorama par votre propre version de logger colorisé, je suis preneur. Pour psutil je n'ai simplement pas réussi à trouver une alternative et l'implémentation des fonctions offertes sont trop compliquées. »

08/03: Rendez-vous: Remarques du Client (Test Hash):

+ Remplacement de Colorama par une classes de couleurs binaires
+ Minimum de librairies
+ Utilisation du wrapper function et pas d'Unittest

15/03: Repousse d'une semaine

Collected logs et comptes rendu

16/03:
Voici quelque points qu'il faudrais corrigé (en plus d'avoir un programme qui fonctionne):
- ajouter un shebang qui permette de lancer python3 directement
- permettre à l'utilisateur (l'étudiant) de passer le nom du programme en paramètre
+ Le retour du test est trop long. Il faudra afficher que le résultat, True, False sans mettre l'expected_output , input

Du 17/03 -> 17/04:
+ Remarque à propos de l'optimisation du code
+ l'insistance sur l'utilsation du fonction wrapper
+ Discussion d'un test d'un très grand fichier (par exemple de taille 10GB)
==> Malheuresement pas envisageable
+ Amelioration globale du code

18/04: Log pour Hash Test:

*deleted unitest, random, logging library
*changed « check(self, expected_hash, resultat) » function from a global function to part of class
*option pour garder les tempfiles -k OK
*remis -t -md5 au debut (simplification de la fonction file_hash qui teste pour md5 ou sha1 selon le besoin)
*ajouter des arguments optionnels pour changer le nombre de fichiers -f, taille de la chaine -c OK
*test pour plusieurs fichiers, même fonction qui permet de tester pour un seul fichier ou plusieurs

Du 19/04 -> 22/05:
+ Discussion sur quel test de TP à faire ==> Client-Serveur
+ Le client a expliquez qu'il faut s'assurer que l'élève à bien fait un fork pour son serveur
==> Multiple Client Test
+ Discussion à propos de la GUI-CUI, comment la lancer et l'utilser, l'importance d'enlever les fichiers .o, executables et faire un clean Make.
+ Choix du type d'execution du programme (Avant-plan vs Arrière-plan)

23/05: Log pour Test Client-Serveur:

Corriger la fonction launch_terminal pour lancer le program silencieusement au background, ou afficher les retours en mode debug

Corriger la fonction too_high_too_low qui prenait le nouveau numéro deviner guess, au lieu de l'ancien

Created Client Test , initialiser avec -c

```
def create_msg(self, correct_number, received_guess):

def server_receive(self, cfd):

def Guessing_Game_Interaction_Test(self, client_socket):

def send_range(self, client_socket):

def intialise_server(self):

TEST_CLIENT_ETUDIANT(self, server_socket)
```

Collected logs et comptes rendu

LOG 05/06/2023: (8h work==> 88h total)

Le programme cible (server/client à tester) s'execute désormais en arrière plan avec subprocess.Popen (>subprocess.run comme elle retourne le PID, qui va nous servir à fermer notre programme et tout ses enfants)

Deleted check_interface_graphique(), les fonctions sont harmonisées pour les GUI et CLI. Deleted terminate_process_by_PORT(), désormais, c'est géré par les signaux (voir bas).

la fonctionn launch_program redirege stdout et stderr > {program_type_name}.txt

Ajout d'un time limit pour chaque test en utilisant socket.settimeout(n_seconds) (10 secondes par Test > pour Multiple Client, c'est 30, pour le serveur c'est 50)

Removed netstat pour check_port_in_use(),malheureusement pas disponible sur toutes les sys expl, use of socket.bind(), gènère un port si c'est in use et l'assigne à la valeur PORT, retourne void .

Implèmenter la function g(f, [bytes]), avec deux versions, direct_version(…, specific_mode), et try_all_version(…), la 1ère faisant appel à la deuxième en except. En effet, c'est pour le handling de la reception d'un seul element (de 4 octets, 2 , ou 1 octets) for LOSS/WIN, au lieu de deux.

Implèmenter unpack_and_determine_mode(self, data, array_of_types) qui utilisent aussi une liste de paramètres et utilisent la recurrence. Elle sert à extraire la première data reçue, l'intervalle de devinage [a,b] et determinent le mode utiliser par le programme cible (le mode correct est 2 octets reçues, les autres sont accèptes et Warning est affiché), retourne le mode dont on va se servir pour le send/receive, et la data unpacked.

values et nombre d'essai sont désormais obligatoire
Ajout de l'argument '-l' :print log directly on screen (au cas où program_log.txt retourne vide par exemple)

Importer signal pour sigterm, dont on a besoin pour envoyer un signal au groupe du programme parent et ses enfants avec os.killpg()

Tester sur l'interface graphique de Debian GNU, Linux, et ses CLI. Tester aussi sur VM Windows 10

525 lignes de codes au total