Rapport Application Informatique: Aider les étudiant-e-s à mieux comprendre les attentes d'un TP

Omar Abdesslem

Juin 2023

Contents

1 Identification des besoins

1.1 Objectif et solutions existantes

- Client : Guillaume Chanel
- Utilisateurs du logiciel : Les étudiants du cours Système d'exploitation
- Objectif principal du logiciel : Permettre aux étudiants de mieux comprendre les attentes de leurs travaux pratiques et d'obtenir une correction indicative instantanée. Le logiciel fournira des tests Python autonomes permettant de lancer le programme de l'étudiant et de le tester rigoureusement selon les critères déterminés par le professeur.
- Solutions existantes :
  - Une des solutions existantes est l'obtention du feedback de l'assistant ou du professeur, ce qui nécessite un temps d'attente supérieur ou égal à 1 jour.
- En quoi la solution que vous proposez est différente?
  La solution proposée ici offre une correction instantanée et autonome des travaux pratiques.
- Licence du code : Licence MIT

1.2 Besoins explicités par le client

- Utilisation minimale de bibliothèques externes (psutil, colorama, etc.).
- Lancement automatique du programme de l'étudiant.
- L'output du programme de l'étudiant doit contenir l'output attendu, pas exactement le même à l'espace près.
- Le programme doit être correctement fermé après la fin du test.
- Le test ne doit pas retourner d'erreurs d'exception, mais doit également imprimer le nom de l'erreur rencontrée et son appartenance à quel sous-test.

1.3 Besoins identifiés par l'étudiant

- Gérer les différences d'exécution du programme selon le système d'exploitation.
- Uniformiser les valeurs par défaut du test et du programme.
- Gérer les états TIME-WAIT et TCP-WAIT qui ne libèrent le port qu'après 30 à 60 secondes.

2 Développement

2.1 Conception

Voici un exemple d'utilisation de mon programme sous forme de scénario :

Supposons qu'un étudiant ait effectué son TP2 sur le sujet "Hash" et souhaite vérifier le bon fonctionnement de son programme, notamment sa capacité à hasher plusieurs fichiers en même temps. Après avoir créé son exécutable avec Make, il peut ouvrir un terminal de commande et saisir la commande suivante :

```
./chemin_vers/test.py -h
```

Cela lui permettra de connaître les différentes options et paramètres à utiliser lors du test. Ensuite, il peut exécuter le test en spécifiant le chemin vers son programme avec d'autres arguments facultatifs :

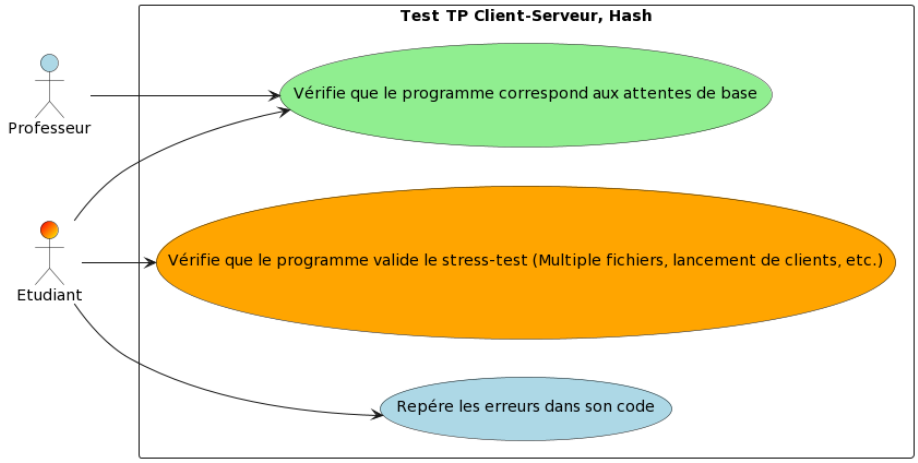
```
./chemin_vers/test.py /chemin_vers/nom_du_programme -other_args
```

Identifions maintenant quelques contraintes que le programme devra satisfaire :

1. Les deux tests créés doivent être exécutables en interface graphique (GUI) et en interface en ligne de commande (CUI) sur les systèmes d'exploitation courants tels que macOS, Linux et ses variantes, ainsi que Windows.

2. Le programme doit être précis, propre (avec une sortie sans trop de lignes inutiles), ne génére pas d'erreurs d'exception, mais utilise plutôt une fonction wrapper pour gérer les exceptions.

2.2 Diagramme de cas d'utilisation



2.3 Interface graphique

Voici les schémas des résultats de l'exécution du programme:

2.4 Sur un GUI

```
parallels@debian-gnu-linux-11:~/Desktop/Parallels Shared Folders/Home/Do
cuments/Test/Hash$ ./test.py ./digest
--- BEGIN TEST ---

--- TESTING STRING_HASH ---
HASH_STRING_COMPARE: SUCCESS
--- TESTING MD5 STRING_HASH ---
HASH_STRING_COMPARE: SUCCESS

--- TESTING FILE_HASH ---
HASH_FILE_COMPARE: SUCCESS
--- TESTING MD5 FILE_HASH ---
HASH_FILE_COMPARE: SUCCESS

--- TESTING MULTIPLE FILE_HASH ---
HASH_FILE_COMPARE: SUCCESS
--- TESTING MD5 MULTIPLE FILE_HASH  ---
HASH_FILE_COMPARE: SUCCESS

le nombre de fichiers est 3, la taille des chaines est 5 octets
```

Test Hash

```
parallels@debian-gnu-linux-11:~/Desktop/Parallels Shared Folders/Home/Do
cuments/Test/Client-Serveur$ ./test.py ./TP-Correct/server 60000 [1,-1,0
,2] 3
Try number changed to:
3 tries
Values changed to:
TOO_HIGH=1, TOO_LOW=-1, WIN=0, LOSE=2
--- BEGIN TEST ---

TRY_LAUNCH_PROGRAM: SUCCESS
TEST_CONNECT: SUCCESS
SINGLE_CLIENT_REQUEST: SUCCESS
TEST_TOO_HIGH_TOO_LOW_LOSE: SUCCESS
MULTIPLE_CLIENTS_TEST_INTERACTIVE: SUCCESS

--- END TEST ---
```

Test Client-Serveur: Serveur

2.5 Sur un CUI

```
parallels@debian-gnu-linux-11:~/Desktop/Parallels Shared Folders
/Home/Documents/Test/Client-Serveur$ ./test.py ./TP-Correct/clie
nt 60000 [1,-1,0,2] 8 -c
Try number changed to:
8 tries
Values changed to:
TOO_HIGH=1, TOO_LOW=-1, WIN=0, LOSE=2
--- BEGIN TEST ---

Port choisi: 60000
TRY_LAUNCH_PROGRAM: SUCCESS
TEST_CLIENT_ETUDIANT: Accepted connection from 127.0.0.1:60616
Test Launch OK
Accepted connection from 127.0.0.1:60618
GUESSING_GAME_INTERACTION_TEST: SEND_RANGE: SUCCESS
Serveur reçoit 1 octet, envoie 2 octets (W/L/.., nombre_à_devine
r)
number to guess: 36
YOU WIN! Guessed 36
Resultat du Guessing Game Interaction Test= SUCCESS
```

Test Client-Serveur: Client

```
parallels@debian-gnu-linux-11:~/Desktop/Parallels Shared Folders
/Home/Documents/Test/Client-Serveur$ ./test.py ./TP-Correct/serv
er 50000 [1,-1,0,2] 3
Try number changed to:
3 tries
Values changed to:
TOO_HIGH=1, TOO_LOW=-1, WIN=0, LOSE=2
--- BEGIN TEST ---

TRY_LAUNCH_PROGRAM: SUCCESS
TEST_CONNECT: SUCCESS
SINGLE_CLIENT_REQUEST: SUCCESS
TEST_TOO_HIGH_TOO_LOW_LOSE: SUCCESS
MULTIPLE_CLIENTS_TEST_INTERACTIVE: SUCCESS

--- END TEST ---
```

Test Client-Serveur: Serveur

3 Implémentation

3.1 Les choix réaliser lors de l'implémentation

- Choix entre des programmes s'exécutant en avant-plan vs arrière-plan.
- Choix de la librairie à utiliser pour l'exécution multiple d'une fonction (Threading).

3.2 Les outils informatiques à disposition

- Les VMs Linux fournies par l'Université.
- Code fourni pour le TP : Client-Serveur.

3.3 Outils informatiques utilisés

- PyCharm, Sublime Text, Gitlab
- Parallels Desktop, VMWare
- StackOverflow, StackExchange
- Python 3.11 Manual, Google

3.4 Pourquoi votre choix s'est porté sur ces outils ?

Les VMs étaient principalement utilisées pour tester mon programme sur différents systèmes d'exploitation. J'ai utilisé Gitlab car c'est la plateforme que nous utilisons dans d'autres cours. J'ai consulté Google, StackOverflow et StackExchange pour obtenir des réponses et des solutions aux problèmes que je rencontrais.

3.5 Tests et évaluation

3.5.1 Quelles méthodes avez-vous utilisées pour tester votre programme ?

J'ai testé mon programme en utilisant les travaux dirigés (TD) développés par moi-même, d'autres étudiants et le professeur.

3.5.2 Quels tests unitaires avez-vous conduits ?

J'ai effectué plusieurs centaines de tests unitaires depuis le début du semestre, dont une dizaine à la fin.

3.5.3 Quels tests d'intégration avez-vous conduits ?

- Lancement avec un programme inexistant, avec un programme incomplet/incorrect..
- Lancement sur Windows, Linux, MacOS.
- Tweaking des paramètres d'entrée.

3.5.4 Quels ont été les critères d'évaluation et de validation du logiciel ?

Pour moi, les critères d'évaluation étaient les suivants :

- Les tests doivent valider les travaux dirigés correctement.
- Les tests ne doivent pas échouer (TimeOut delay).
- Les tests doivent capturer toutes les exceptions et les afficher dans la partie correspondante des sous-tests.
- Les tests doivent guider l'étudiant vers son erreur en fournissant des indications (par exemple : la taille doit être de 2 octets, pas 4 ; le serveur doit recevoir le WIN/LOSE avant le guess, etc.).

3.5.5 Le logiciel a-t-il passé l'évaluation ?

Dans une certaine mesure, oui.

3.5.6 Quelles sont les pistes pour améliorer le logiciel ?

- Optimisation du code.
- Vérification des processus zombies du TP : Client-Serveur.
- Fournir plus de détails pour les erreurs.

4 Organisation

4.1 Comment avez-vous divisé le travail entre le client et l'équipe de développement ?

J'étais seul.

4.2 Comment avez-vous organisé vos réunions avec l'enseignant encadrant et le client ?

J'ai assuré le suivi du travail à effectuer par des réunions avec mon enseignant encadrant/client. Ces réunions ont eu lieu de manière hebdomadaire tout au long du semestre, sauf pendant les vacances, à l'exception des dates du 3 mai et du 15 mars.

4.3 Quel volume horaire de travail avez-vous fourni ?

Sans le rapport, j'ai fourni 88 heures de travail, et avec le rapport, 94 heures.

5 Formation

Décrivez comment vous avez utilisé les acquis de votre formation pour résoudre la commande de votre client. Quels enseignements vous ont été utiles ?

Les cours les plus utiles pour mon projet étaient la Programmation Orientée Objet par Jean-Luc Falcone, qui m'a servi à créer des classes et des méthodes, et le cours Système d'exploitation par Guillaume Chanel, qui m'a aidé à comprendre la communication par paquets, la création de sockets et les signaux.

Décrivez les notions, principes et techniques que vous avez dû acquérir durant le projet, car ils ne faisaient pas partie de votre formation initiale.

- Globalement, la création de tests en Python.
- L'utilisation d'une fonction wrapper qui capture les exceptions.
- Apprendre à utiliser plusieurs bibliothèques en Python, notamment : import argparse, subprocess, time, os, threading, signal.

Pensez-vous qu'il soit possible ou utile de les inclure dans les deux premières années du bachelor ?

Il serait utile d'inclure la création de tests dans une partie d'un cours de Bachelors, par exemple, CPOO, mais c'est aussi une notion que tous les ingénieurs finissent par apprendre, que ce soit à l'université ou au travail.

5.1 Feedback

En général, tout s'est bien passé. Pas de critiques à mentionner.

6 Annexes et GIT

Le code et la documentation utilisateur indiquant comment utiliser votre programme est contenu dans la ReadMe du GIT .