Livrable intermédiaire 1:
Projet ChatAmu
Application Réseaux

ABDEL WAHAB Ismail

18 Mars 2020

Contents

1 Présentation des ressources **2**
- 1.1 Fichiers liés à git . . . 2
- 1.2 Où trouver les sources des livrables . . . 2
- 1.3 Hiérarchie du code . . . 2
- 1.4 Organisation des tests . . . 2

2 Choix des Langages **3**
- 2.1 Langage principal de programmation du projet . . . 3
- 2.2 Ouverture aux autres langages . . . 3
- 2.3 Langages utilisés pour les test . . . 3

3 Gestion des Clients et des Serveurs **3**
- 3.1 Les Clients . . . 3
  - 3.1.1 La concurence . . . 3
  - 3.1.2 Les Erreurs . . . 3
- 3.2 Les Serveurs . . . 4
  - 3.2.1 La communication asynchrone . . . 4

4 Mise en oeuvre des tests **4**
- 4.1 Comment executer le test pour Chatamu . . . 4
- 4.2 Les Librarie utilisées . . . 4
- 4.3 Fonctionnement du test . . . 4

1 Présentation des ressources

1.1 Fichiers liés à git

Ce projet est en privé sur mon git personnel, d'où la présence des fichiers suivants, cela me permet de sauvegarder mon travail et de pouvoir travailler sur differentes machines si je n'ai pas un accès directe à mon poste:

```
.git/      |---
    ...    | Fichiers liés au
.gitignore |     repo git
README.md  |---
```

1.2 Où trouver les sources des livrables

Le dossier **REPORTS/** regroupe tout les éléments qui ont été utilisés pour rédiger les livrables:

```
REPORTS/
  tex_files/
     *.tex                --> Fichiers LaTeX
  Livrable_[X]-ChatAmu-ABDELWAHAB_Ismail.pdf --> Livrable [X]
```

1.3 Hiérarchie du code

Dans **src/**, vous est fourni le code du projet, délimité pour l'instant en serveurs et clients:

```
src/
  Clients/
    client.sh            --> Script bash lance un client spécifié
    SimpleClient.java    --> Client de base (LOGIN + MESSAGE)
    *.class              --> Fichiers compilés java
  Servers/
    server.sh            --> Script bash lance un serveur spécifié
    SalonCentral.java    --> Server Centralisé
    ChatamuCentral.java  --> Server de chat Centralisé
```

1.4 Organisation des tests

Tout les ressources sont dans **tests/**. A noter que tout les tests sont automatiques et vont chercher les scripts et les fichiers dont ils ont besoin d'eux même.

```
tests/
 txt_sources/
    [client_name].txt   --> texte dit par le client [client_name]
 chatamuTest.py         --> Script de test du serveur ChatamuServer
```

2 Choix des Langages

2.1 Langage principal de programmation du projet

Le choix du langage c'est fait assez rapidement. En effet durant nos Travaux Pratiques nous avons travaillé en Java, il est donc naturel de réutiliser à notre avantage ce que nous avons appris.

De plus, j'utilise l'IDE Intellij depuis le debut de ma deuxième année de licence, il est donc assez confortable de coder et debogger du Java ce cette manière, car j'en ai pris l'habitude.

2.2 Ouverture aux autres langages

Selon l'avancement du projet, il est possible que je devellope, un client ou un seveur en Python ou même en C.

2.3 Langages utilisés pour les tests

C'est après quelques difficultés, et surtout grâce au report des dates de rendu, que j'ai pu m'initier au librairies "subprocess", "os" et "threading" de python.

Ce qui m'as permit de réaliser un script de test automatique pour le serveur Chatamu, ce script utilise également d'autres scripts bash afin de simplifier, le lancement des serveurs et clients.

3 Gestion des Clients et des Serveurs

3.1 Les Clients

3.1.1 La concurence

La première difficulté rencontrée, à été l'obligation de gérer simultanément les écritures du client et l'affichage du client (les messages reçus).

Après une courte tentative en vain, de client utilisant des selecteurs, il m'as parrut beaucoup plus efficace de s'occuper directement de l'ecriture du client dans le cours d'execution courrant, et de simplement créer un thread afin d'administrer les messages reçus pour les afficher.

C'est donc en multi-thread que nous pouvons, de manière concurrente, s'occuper des E/S du client.

3.1.2 Les Erreurs

Le premier message d'un client doit être un "LOGIN pseudo", si ce n'est pas le cas, le serveur notifie le client qui a tenté de se connecter, puis ferme la connection.

Le même choix de rejet de connection à été fait en cas d'erreur sur le protocol "MESSAGE". L'argument étant de dire que le client n'as pas utilisé un protocol certifié par le serveur et que par defaut, ce dernier refuse toutes communications par la suite.

3.2 Les Serveurs

3.2.1 La communication asynchrone

Ici notre serveur doit être capable de recevoir et d'envoyer des messages de facon asynchrone, si ce n'était pas le cas, un client pourrait attendre plusieurs secondes, voire minutes, pour recevoir les messages du serveur.

Pour palier à cela, on utilise un pool de thread dynamique. Le thread principal génère un thread fils pour chaque nouvelle connexion.

Ce thread fils généré va s'occuper de:

1. L'autentification du client.
2. En cas de reussite : Créer un thread fils qui s'occupera des écritures du client
3. Afficher les messages reçus du serveur.

Ainsi nous avons le thread main qui s'occupe du nouveau client. Et pour chaque client deux threads supplémentaires qui s'occupent de l'ecriture et de la lecture de leurs données.

4 Mise en oeuvre des tests

4.1 Comment executer le test pour Chatamu

A l'aide d'un terminal placez vous dans le dossier **test/** puis saisir:

```
./chatamuTest.py
    (ou alors)
python3 chatamuTest.py
```

4.2 Les Librarie utilisées

Voici la liste des librairies qui ont été utilisées pour ce test:

```
os  -  subprocess  -  time
time  -  threading  - socket
```

Note: Ici la librairie "socket" est utilisée pour simuler un client netcat.

4.3 Fonctionnement du test

Le test lance le serveur Chatamu à l'aide du script pm_daemonize.sh précédement fournis.

Puis lance un SimpleClient à l'aide d'un thread, ce dernier représente un client, nommé JAK. Il se connectera, puis enverra quelques messages, et finira par un message ne respectant pas la typo attendue par le serveur. Ce qui montrera par la suite qu'un client est bien déconnecté si il n'utilise pas un bon protocol.

En même temps, un deuxième thread va lancer le client netcat. Ce client est simulé par l'utilisation de la librairie socket, j'ai délibérement fais ce choix car cela me permet d'envoyer un message par seconde.

C'est par ce moyen que le client netcat va voir les messages de JAK et sa deconnection aussi.

Le client netcat finira aussi sur une erreur de protocol, ce qui permettra de verifier que le serveur avertit bien les clients en cas d'erreur.