Rapport de projet : Chat – édition processus & pipe

Daniel Defoing Ibrahim Dogan Abdalrahman El Hussein

November 21, 2024

Contents

1 Introduction

Ce rapport décrit la conception du premier projet dans le cadre du cours de Systèmes d'Exploitation (INFO-F201). Il présente les choix de conception qui ont guidé notre développement, et les difficultés qui ont pu survenir durant celui-ci. Pour voir de façon précise les changements ayant eu lieu tout au long du projet et la contribution de chacun, veuillez consulter le repository GitHub de notre projet.

2 Choix du langage : pourquoi C++ plutôt que C ?

Une raison absolument fondamentale qui a guidé cette décision est que C++ possède des éléments absents en C tels que les références, les strings ou encore les classes. Or, nous avons considéré que ces éléments pourraient :

- **Faciliter notre développement** en enrichissant la boîte à outils mise à notre disposition d'outils pratiques pour développer des applications en général, comme les classes.
- Éviter de devoir réinventer la roue en pouvant **abstraire ou simplifier certaines opérations** grâce aux outils du C++. Par exemple, nous avons pu dans certains cas utiliser des références ou pointeurs intelligents pour résoudre certains problèmes simplement, ce qui se serait avéré plus compliqué en C, langage qui ne possède pas ces outils.
- éventuellement : dire qu'on a plus d'expérience avec C++ au global, avec le cours de LDP1, etc

Ceci montre que nous avons trouvé des avantages à utiliser C++ plutôt que C, mais il faut également noter que nous n'avons pas vu d'avantage majeur à faire du C plutôt que du C++. Ce choix du langage s'est donc naturellement imposé à nous.

3 Principes généraux de conception

Cette section discute des grands principes qui ont guidé notre développement, des idées qui ont défini la façon dont nous avons écrit du code. Cette section n'est bien sûr pas exhaustive, les principes mentionnés ci-dessous s'appuient sur les principes généraux de codage qui ont été appris depuis le début du bachelier. La gestion d'éléments spécifiques à l'application (signaux, mémoire partagée...) sera discutée dans la section suivante.

Nos deux principes de base furent donc :

3.1 Abstraire des opérations “de bas niveau” le plus possible

Nous avons tenu à masquer le fonctionnement interne de notre application le plus possible, afin de permettre à l'application d'éventuellement évoluer (par exemple en une application de chat en ligne). C'est évidemment un des fondements du principe d'encapsulation [source]. Il suffit de voir des noms de méthodes comme `send_message` ou `open_sending_channel` pour s'en convaincre [...]

3.2 Montrer un maximum d'informations à l'utilisateur, au-delà des erreurs

Nous avons mis en place un système de warnings, en plus des erreurs vraiment critiques : le but de ceci était d'être le plus explicite possible pour l'utilisateur, et de proposer une séparation entre les erreurs dites critiques et les autres, moins graves.

4 Choix d'implémentation

BLABLA TODO

5 Difficultés rencontrées et solutions

Pour clore ce rapport, nous allons discuter des difficultés rencontrées pendant le développement sur l'implémentation des points de la section précédente, ainsi que sur les difficultés à suivre nos principes généraux de conception.

5.1 Gestion de la communication inter-processus

Envoyer des messages exactement de la bonne taille et au bon moment n'a pas été une tâche facile. Les concepts derrière ceci n'étaient pas forcément compliqués à comprendre, mais c'est leur combinaison et le grand nombre d'erreurs possibles qui a posé problème.

5.2 Signaux et échec d'opérations primitives (comme `write`)

Là encore, la tâche semblait facile de prime abord : vérifier les valeurs de retour de ces opérations primitives, et en cas de valeur problématique, gérer l'erreur proprement. La difficulté n'a pas été de vérifier les valeurs de retour, mais d'afficher des erreurs et quitter le programme en évitant le code spaghetti (donc : ne pas mettre des exit partout, mais gérer les choses proprement via des conditions, regroupements pertinents de code, etc).

La plupart de ces difficultés rencontrées ont un lien entre elles, dans le sens où la difficulté de la tâche ne réside pas dans les concepts derrière celle-ci, mais plutôt dans une gestion exhaustive des cas possibles tout en conservant une bonne clarté de code.