Rapport de projet: Chat – édition processus & pipe

Daniel Defoing (`ddef0003`)) Ibrahim Dogan (`ULBID`))
Abdalrahman El Hussein (`ULBID`))

28 novembre 2024

Table des matières

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 fondamentale qui a guidé cette décision est que C++ possède des fonctionnalités absentes en C tels que les références, les chaînes de caractères (ou strings) ou encore les classes. Ces éléments apportent des avantages majeurs :

- Ils facilitent le développement en enrichissant la panoplie mise à disposition des développeurs d'outils pratiques pour développer des applications en général, comme les classes.
- Ils évitent aux développeurs de devoir réinventer la roue en pouvant abstraire ou simplifier certaines opérations grâce aux outils susmentionnés du C++. Par exemple, l'utilisation des références ou classes a été importante pour résoudre certains problèmes rapidement, chose qui se serait avérée plus compliquée en C qui ne possède pas ces outils.

Ceci montre que des avantages ont été trouvés à utiliser C++ plutôt que C, alors qu'aucun avantage majeur n'a été identifié en faveur de l'utilisation de C par rapport à C++. Ce choix du langage s'est donc naturellement imposé comme le plus adapté.

3 Principes généraux de conception

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

3.1 Abstraction des opérations "de bas niveau" au maximum

On a cherché 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 l'un des fondements du principe d'encapsulation, qui favorise une séparation claire entre l'interface et l'implémentation interne d'un module [1]. Il suffit de voir des noms de méthodes comme `send_message` ou `open_sending_channel` pour s'en convaincre : ces noms disent simplement ce que la méthode *fait*, pas *comment elle le fait*.

3.2 Informer l'utilisateur tant que possible

En plus des erreurs critiques censées stopper l'exécution du programme, un système de warnings a été mis en place. Le but de ceci était d'être le plus explicite possible pour l'utilisateur sur l'état de l'application, et de proposer une séparation entre les erreurs dites *critiques* et les autres, moins graves.

4 Choix et difficultés d'implémentation

4.1 Makefile

Le makefile fourni au commencement du projet a été repensé sous plusieurs angles pour permettre une meilleure robustesse avec des features basiques mais pratiques, dont voici des exemples :

- Création automatique d'un dossier `obj` pour stocker les fichiers objet s'il n'existait pas auparavant
- Réorganisation des variables du makefile pour rendre les commandes de compilation plus flexibles, plus génériques. En d'autres termes, il suffit de modifier la valeur d'une variable pour adapter les options de compilation.

Cette partie n'a pas posé de difficultés particulières.

4.2 Gestion des exceptions et valeurs de retour

Une spécificité de ce projet est qu'il implique l'utilisation d'un grand nombre de fonctions système prédéfinies (`write` ou `read` notamment). Ces fonctions, construites à la base pour le langage C, ne lèvent pas d'exception (absentes en C), mais ont plutôt des valeurs de retour qui indiquent l'issue de l'opération exécutée. Par exemple, un appel à `write` retournera la valeur -1 si une erreur devait arriver d'une façon ou d'une

autre.

Difficulté rencontrée : Une conséquence de ce fonctionnement des fonctions système en C est que, contrairement à une exception, une valeur de retour indiquant une erreur n'interrompt pas le programme en cours. Cela signifie que, si l'on n'est pas assez rigoureux sur la vérification des valeurs de retour, on risque d'avoir des erreurs "silencieuses" en causant d'autres, et cela peut mener à des erreurs très difficiles à débugger. Une difficulté rencontrée lors du développement a donc été de gérer proprement toutes ces opérations prédéfinies dès le début, et d'éviter des conditions à tout va pour éviter de se retrouver avec un code désordonné, difficile à lire et à maintenir [2].

Résolution : afin de faciliter la maintenabilité du code et de minimiser les oublis, il a été décidé de centraliser autant que possible la gestion des erreurs dans une classe,`ExceptionHandler`, qui a permis une uniformisation de la gestion des erreurs dans le programme, ainsi qu'à un raffinement des types d'erreurs que notre programme détecte (via un système de warnings, mentionné plus haut).

4.3 Programme `chat`

4.3.1 Choix d'implémentation

La logique à suivre pour l'implémentation des canaux de communication entre deux utilisateurs semblait assez évidente, car mentionnée en travaux pratiques [2] : il fallait d'abord initialiser les canaux de communication (les pipes, donc), et ensuite créer un processus fils (via un appel à `fork()`) pour avoir un processus *écrivain* de messages (via les pipes) et un processus *lecteur*.

4.3.2 Mode `bot`

pas de texte fourni TODO

4.3.3 Mode `manuel`

pas de texte fourni TODO

4.3.4 Mémoire partagée pour le mode `manuel` (INCOMPLET, TODO)

Choix d'implémentation : Pour créer de la mémoire partagée entre les différents processus ayant un lien de parenté, nous avons décidé d'utiliser `mmap`, un choix qui semble évident : pour une mémoire partagée telle que demandée dans le projet, les

fonctionnalités de `mmap` suffisaient amplement, là où utiliser une autre librairie comme `shm` se serait avéré plus complexe sans gain notable.

Pour la taille de la mémoire partagée, nous avons pris la taille maximale du "buffer" dans lequel les messages sont stockés lorsque le mode `manuel` est activé. Comme indiqué dans les consignes du projet, les messages en attente sont affichés dans 3 différents cas : si le signal SIGINT est reçu, si un message est envoyé par l'utilisateur ou si plus de 4096 octets sont en attente d'être affichés. Nous avons donc décidé d'afficher tous les messages et de vider le "buffer" lorsqu'une variable nommée `total_chars` dépasse la valeur 4096.

Difficulté rencontrée : comme mentionné auparavant, `mmap` a été utilisé pour initialiser une mémoire partagée. Pour stocker les messages dans la mémoire partagée, l'utilisation de `std::queue` avait été décidée initialement. Cependant, nous avons réalisé que ce n'était pas fonctionnel. En effet, les éléments d'une structure de données STL peuvent en réalité être des pointeurs qui pointent vers une autre zone mémoire. Cela créait donc des crashs du programme.
Résolution (incomplet) : Par après, nous avons résolu ce problème en stockant les messages dans un seul "buffer".

5 Conclusion : la difficulté majeure de ce projet

Les difficultés spécifiques à chaque partie du programme ont été discutées en profondeur : il est temps de prendre une vue de plus haut niveau sur l'ensemble du projet, pour discuter de sa difficulté principale.

La plupart de ces difficultés rencontrées ont en fait une origine commune, dans le sens où la difficulté de la tâche ne réside pas dans les concepts derrière celle-ci, mais plutôt dans leur cohabitation au sein d'un même programme. Cela impliquait une gestion exhaustive des cas possibles tout en maintenant une clarté et une structure de code rigoureuses.

6 Bibliographie

Sources consultées pour la dernière fois le 28 Novembre 2024, 16h31.

1. *Encapsulation (programmation)* - Wikipédia en français
2. TPs 3 à 5 du cours de Systèmes d'Exploitation (INFO-F201)