

Rapport de Projet d'OS 1

David de Meester, Imrane Benaissa, Maxime Lauzeral

21 novembre 2024

1 Contenu

1. Introduction et présentation
2. Construction du programme
 - 2.1. Dépendances
 - 2.2. Installation
3. Liste des fonctionnalités
4. Démonstration en pratique
 - 4.1. Usage
 - 4.2. Illustrations
 - i. Utilisateur bob exécute ./chat
 - ii. Bob veut communiquer avec alice
 - iii. Alice veut communiquer avec bob
 - iv. Alice envoie un message à bob
 - v. Bob envoie un message à alice
 - vi. Alice stoppe la conversation
5. Difficultés rencontrées, solutions apportées, choix d'implémentation
 - 5.1. Préambule et balbutiements
 - 5.2. Pipes, signaux et processus
 - 5.3. Mémoire partagée et bot
6. Conclusion

2 Introduction et présentation

Ce projet de **INFO-F201** consiste en un petit programme permettant de discuter avec une personne ou un robot localement à l'aide de pipes nommés. Nous allons d'abord expliquer comment installer le programme, puis lister les fonctionnalités de ce dernier et montrer des exemples d'utilisation. Ensuite, nous discuterons des difficultés rencontrées et des solutions apportées en justifiant nos choix d'implémentation. Enfin, nous terminerons par une conclusion donnant notre sentiment global sur le projet.

3 Construction du programme

3.1 Dépendances

Le programme a été développé pour fonctionner sur une distribution Debian. Quelques préliminaires sont requis pour assurer une bonne installation. Ouvrez le terminal (**Ctrl+Alt+T**) et exécutez les commandes ci-dessous :

```
sudo apt update
sudo apt install build-essential
sudo apt install g++
sudo apt install make
```

3.2 Installation

Nous avons décidé de créer un dépôt GitHub pour faciliter la mise en œuvre du projet. Comme les fichiers sources et un **Makefile** se trouvent dans le zip, vous pouvez compiler et exécuter le programme depuis le chemin de ce dernier à l'aide de **make**. Alternativement, si vous perdez le fichier zip, vous pouvez exécuter les commandes suivantes dans le répertoire de votre choix :

```
git clone https://github.com/kuzaiko/OS_PROJECT_1
cd OS_PROJECT_1
make
```

4 Liste des fonctionnalités

Le programme sans option permet de discuter avec un interlocuteur, de recevoir des messages et d'en envoyer. Cependant, il est possible d'ajouter des options pour étendre les fonctionnalités du programme. Les options supplémentaires proposées sont les suivantes :

```
#permet de recevoir les messages de l'interlocuteur uniquement sous certaines conditions
--manuel
```

```
#permet de discuter avec un bot et d'exécuter les commandes ci dessous sensible à la casse
--bot
```

« **liste** » : lister tous les fichiers du dossier de travail du robot ;
 « **li FICHIER** » : lire le contenu du fichier « FICHIER » ou afficher une erreur en cas d'échec (p.ex., si le fichier est inexistant) ;
 « **qui suis-je** » : donner le pseudonyme du destinataire ;
 « **au revoir** » : terminer le robot avec le code de retour 0.

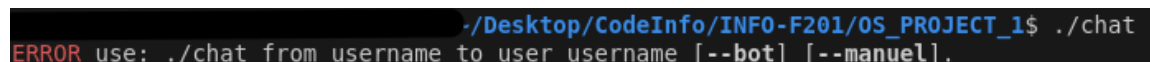
5 Démonstration en pratique

5.1 Usage

Dans le répertoire où se trouve le fichier exécutable, exécutez :

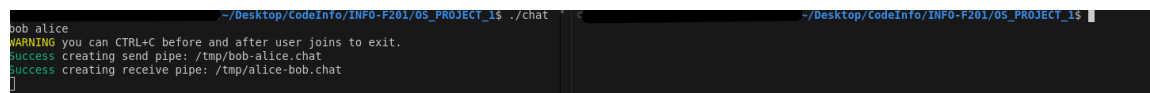
```
./chat
```

5.2 Illustrations



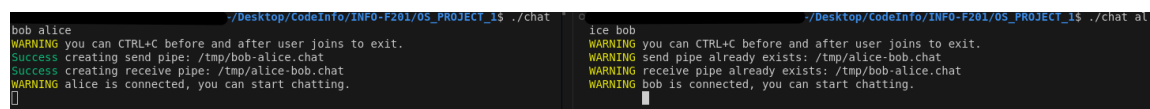
```
~/Desktop/CodeInfo/INFO-F201/OS_PROJECT_1$ ./chat
ERROR use: ./chat from username to user username [--bot] [--manuel].
```

FIGURE 1 – Utilisateur bob exécute ./chat



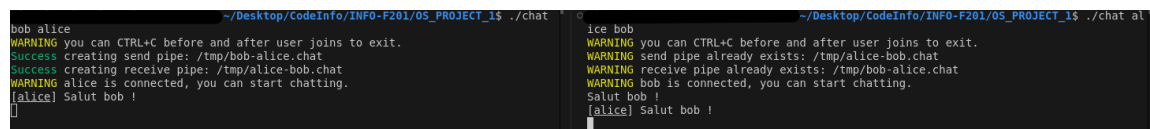
```
~/Desktop/CodeInfo/INFO-F201/OS_PROJECT_1$ ./chat
bob alice
WARNING you can CTRL+C before and after user joins to exit.
Success creating send pipe: /tmp/bob-alice.chat
Success creating receive pipe: /tmp/alice-bob.chat
```

FIGURE 2 – Bob veut communiquer avec alice



```
~/Desktop/CodeInfo/INFO-F201/OS_PROJECT_1$ ./chat
alice bob
WARNING you can CTRL+C before and after user joins to exit.
WARNING send pipe already exists: /tmp/alice-bob.chat
WARNING receive pipe already exists: /tmp/bob-alice.chat
WARNING bob is connected, you can start chatting.
```

FIGURE 3 – Alice veut communiquer avec bob



```
~/Desktop/CodeInfo/INFO-F201/OS_PROJECT_1$ ./chat
alice bob
WARNING you can CTRL+C before and after user joins to exit.
Success creating send pipe: /tmp/bob-alice.chat
Success creating receive pipe: /tmp/alice-bob.chat
WARNING alice is connected, you can start chatting.
[alice] Salut bob !
```

FIGURE 4 – Alice envoie un message à bob

```

bob alice ~/Desktop/CodeInfo/INFO-F201/OS_PROJECT_15 ./chat
WARNING you can CTRL+C before and after user joins to exit.
Success creating send pipe: /tmp/bob-alice.chat
Success creating receive pipe: /tmp/alice-bob.chat
WARNING alice is connected, you can start chatting.
[alice] Salut bob !
[alice] Salut Alice !
[alice] Salut Alice !

ice bob ~/Desktop/CodeInfo/INFO-F201/OS_PROJECT_15 ./chat al
WARNING you can CTRL+C before and after user joins to exit.
WARNING send pipe already exists: /tmp/alice-bob.chat
WARNING receive pipe already exists: /tmp/bob-alice.chat
WARNING bob is connected, you can start chatting.
[alice] Salut bob !
[alice] Salut Alice !
[alice] Salut Alice !

```

FIGURE 5 – Bob envoie un message à alice

```

bob alice ~/Desktop/CodeInfo/INFO-F201/OS_PROJECT_15 ./chat
WARNING you can CTRL+C before and after user joins to exit.
Success creating send pipe: /tmp/bob-alice.chat
Success creating receive pipe: /tmp/alice-bob.chat
WARNING alice is connected, you can start chatting.
[alice] Salut bob !
[alice] Salut Alice !
[alice] Salut Alice !
[alice] Je dois m'en aller :(
WARNING alice disconnected, closing chat.
Success removing send pipe: /tmp/bob-alice.chat
Success removing receive pipe: /tmp/alice-bob.chat
~/Desktop/CodeInfo/INFO-F201/OS_PROJECT_15

ice bob ~/Desktop/CodeInfo/INFO-F201/OS_PROJECT_15 ./chat al
WARNING you can CTRL+C before and after user joins to exit.
WARNING send pipe already exists: /tmp/alice-bob.chat
WARNING receive pipe already exists: /tmp/bob-alice.chat
WARNING bob is connected, you can start chatting.
[alice] Salut bob !
[alice] Salut Alice !
[alice] Salut Alice !
[alice] Je dois m'en aller :(
[alice] Je dois m'en aller :(
^C
WARNING signal SIGINT received, closing chat.
~/Desktop/CodeInfo/INFO-F201/OS_PROJECT_15

```

FIGURE 6 – Alice stoppe la conversation

6 Difficultés rencontrées, solutions apportées, choix d'implémentation

6.1 Préambule et balbutiements

L'utilisation de C n'était pas possible en raison de l'utilisation de certaines bibliothèques comme `<iostream>`, nous avons donc choisi d'utiliser C++. La gestion des erreurs s'est faite via des macros dans un fichier header pour permettre une meilleure lisibilité. Le soucis d'organisation nous a également préoccupés; nous avons opté pour une structure modulaire séparant les fonctions, les classes et le `main` dans différents fichiers et créé un dossier supplémentaire contenant les différents headers. Nous avons trouvé l'idée intéressante de créer certaines variables globales pour faciliter la communication entre les processus et d'également les mettre dans un header.

6.2 Pipes, signaux et processus

Après avoir trouvé un début de structure, nous avons commencé à travailler sur les erreurs d'arguments et les pipes nommés. Nous avons trouvé élégant de créer une classe pour gérer la création, l'ouverture et la suppression des pipes. Nous avons essayé au maximum de gérer les erreurs et de les afficher pour permettre à l'utilisateur de comprendre ce qui ne va pas. De plus, nous avons utilisé des signaux pour gérer les interruptions de l'utilisateur et la déconnexion de l'interlocuteur. Dans le premier cas, nous avons géré `SIGINT` dont le processus père s'occupe et dans le second cas, nous avons utilisé un `SIGPIPE` que le processus enfant appelle pour informer le père de la déconnexion de l'interlocuteur. Nous avons fait attention de fermer les pipes avant de les supprimer.

6.3 Mémoire partagée et bot

À la suite de la création de pipes, de la gestion des signaux et des processus, nous sommes passés au développement du mode manuel et du bot. Nous avons utilisé `shm_open` pour permettre

le partage de mémoire entre les processus et avons choisi d'utiliser 4096 bytes pour la quantité de mémoire partagée. Tout simplement car dans le mode manuel, nous avons une limite imposée de 4096 bytes avant l'envoi automatique des messages stockés dans la mémoire partagée. Pour ce qui est des 3 scénarios du mode manuel, seul le processus enfant affiche le cas où la limite est atteinte. Dans les deux autres cas, le père s'occupe de l'affichage. (SIGINT reçu et envoi de message par l'utilisateur) Nous avons géré le cas d'une fermeture de stdin (CTRL+D) pour permettre à l'utilisateur de quitter le programme. Encore une fois, nous avons fait attention à la gestion des erreurs et à la bonne fermeture des pipes et de la mémoire partagée.

À ce stade, il ne restait plus qu'à implémenter le bot. Nous avons créé un script **Bash** permettant de simuler un bot et de répondre aux commandes spécifiques. Nous avons utilisé **coproc** comme conseillé dans le pdf, afin de pouvoir gérer les entrées et sorties du programme chat. Enfin, nous avons veillé à utiliser **flush** sur la sortie standard pour éviter les problèmes de buffering.

7 Conclusion

En conclusion, ce projet nous a permis de mettre en pratique les notions vues en cours, de nous familiariser avec les pipes nommés, les signaux, la gestion des processus, la mémoire partagée et le scripting bash. Nous avons rencontré des difficultés importantes quant à l'organisation du code et du travail d'équipe. Nous sommes parti dans une direction et avons fait des choix d'implémentation qui nous l'espérons, sont pertinents. Le projet était très intéressant et le pdf de consignes était très détaillé, nous avons fait en sorte de le suivre au mieux.