Info F-201 : Projet d'OS
Rapport

Auteurs: Liefferinckx Romain, Rocca Manuel, Radu-Loghin Rares
Matricules: 000591790, 000596086, 000590079
Section: INFO

2024, 10 Novembre

Contents

1 Introduction

1.1 Présentation du projet et contexte

Dans le cadre de notre cours d'OS, nous avons réalisé un projet qui consiste à implémenter un chat en C. Ce chat, permet la communication entre deux utilisateurs via deux terminaux différents sur un même ordinateur grâce à des pipes nommés pour la transmission de messages. Le chat est composé de deux parties, celle décrite ci-dessus et une autre écrite en bash, faisant office de chat-bot. Ce chat-bot est conçu pour simuler un utilisateur en répondant automatiquement à des commandes spécifiques envoyées par l'interlocuteur.
Le projet se compose donc de deux parties : le programme de chat (chat) et le script Bash (chat-bot).

1.2 Objectifs du projet

L'objectif de ce projet est de mettre en pratique les concepts vus en cours d'OS, notamment la gestion des processus, des signaux, la gestion de la mémoire partagée, et la communication inter-processus. Ce rapport décrit les choix d'implémentation, les difficultés rencontrées et les solutions mises en œuvre utilisée dans la construction de ce projet.

2 Choix d'Implémentation

2.1 Choix du langage

Dans le cadre de ce projet, nous avions le choix entre le C et le C++ comme langage de programmation. Nous avons fait le choix d'utiliser du C car c'est en C que nous avons vu la matère durant les séances de travaux pratiques. De plus, les OS modernes sont majoritairement écrits en C, bien qu'ils utilisent un peu de C++.

2.2 Gestion des processus

2.3 Gestion des signaux

Pour la gestion des processus, nous avons décidé d'utiliser "sigaction" et non pas "signal" car celle-ci nous a été recommandée par notre assistant durant un Tp. En effet, "signal" est une fonction qui ne permet pas de gérer les signaux de manière fiable, du moins nous n'avons pas réussi a la faire fonctionner correctement dans notre projet et avons changé. "sigaction" est donc la fonction à utiliser pour gérer les signaux grâce à sa structure "struct sigaction" qui permet une certaine modularité de la structure du code et dans la gestion des signaux.

2.4 Gestion de la mémoire partagée

La mémoire partagée, utilisée par le mode "–manuel", est implémentée à l'aide d'une structure "sharedMemo". Celle-ci permet une gestion par indexage de la mémoire, sans avoir à se soucier des pointeurs. Elle est donc composée d'un entier reprenant le décalage et d'un tableau de 4096 octets, représentant la mémoire elle-même.
En ce qui concerne son fonctionnement, nous sommes partis sur la structure de donnée "queue". Elle permet de récupérer facilement le premier string entré grâce au principe "first-in, first-out". Nous retrouvons les fonctions pour lire et écrire, ainsi que celles pour initialiser la mémoire et la désallouer. Une fonction permettant la lecture complète la mémoire

est également mais non-utilisée. Finalement, chaque étape critique de la vie de la mémoire partagée est complémentée par de la gestion d'erreurs pour s'assurer de son bon fonctionnement.

2.5 Communication inter-processus

3 Difficultés Rencontrées et Solutions

3.1 Première difficulté: SIGINT et SIGPIPE

Pour nous, la première difficulté fut celle de la gestion des signaux avec le "SIGINT" et le "SIGPIPE" qui sont envoyés par le système d'exploitation. En effet, le "SIGINT" est envoyé lorsqu'on appuie sur "Ctrl+C" et le "SIGPIPE" lorsque le pipe est fermé. De plus, quand un signal est intercepté, les deux ./chat doivent être fermé, c'est la que tout s'est compliqué. Pour gérer tout ca, nous avons décidé d'utiliser "sigaction" plutôt que "signal" comme expliqué dans la section précédente. Les signaux sont gérés dans la main, qui utilise une fonction signal_management pour les gérer, celle-ci est appelée à chaque fois qu'un signal est reçu, et ferme l'entrée standard, les descripteurs de fichiers sont bien fermé après chaque utilisation, les processus sont bien tués et les fifo bien unlink.

3.2 Deuxième difficulté: Mémoire partagée

La seconde difficulté que nous avons rencontrée concerne quant à elle la façon d'implémenter la mémoire partagée. Premièrement, nous avons décidé d'utiliser la structure "sharedMemo", expliquée plus haut, pour éviter les complications que peut apporter l'utilisation de l'arithmétique de pointeurs. Ensuite vient le choix de la structure de donnée abstraite à utiliser pour que le fonctionnement de la mémoire corresponde à nos besoin. Nous avons d'abord opté pour le "stack" et son principe de "last-in, first-out", mais comme nous cherchons à récupérer le premier string entré, la "queue" tombait sous le sens. Pour expliquer brièvement notre manière de procéder, nous avons une boucle qui, lorsqu'un nouveau string est ajouté, décale tous les mots dans la mémoire vers la droite (vers la fin de la mémoire). Il est donc placé au début de notre espace alloué. En ce qui concerne le retrait du premier string entré, nous procédons avec un pop classique. Nous récupérons le dernier élément, sans oublier de placer l'index à la fin du string juste derrière celui retiré.

3.3 Troisième difficultée: Taille du buffer

Pour la taille du buffer, nous avions mis 256 octets, ce qui nous semblait correcte mais nous nous sommes rendu compte en posant la question à notre assistant que le buffer devait avoir une taille infinie, cela quelque jours avant la remise du projet. Nous avons donc du changer cela en vitesse. Pour la résolution de ce problème, nous avons utilisé des concepts vu lors du premier Tp d'introduction ou nous devions agrandir la taille d'une liste en la doublant tout en utilisant malloc pour allouer la mémoire afin que celle-ci ne sépuise pas. De plus, évidemmment, un changement de ce genre en dernière minute a des conséquences sur le reste du code, cela a provoqué des bugs que nous avons du corriger.

4 Solutions Originales et Améliorations

4.1 Gestion des variables globales

Lors de la création du projet, étant donné que nous avons choisit le C comme langage de programmation, il était interdit d'utiliser de l'orienté objet et donc aucune variable dans les instanciations des objets. Le premier réflexe est donc de mettre "const <nom de la variable>= "valeur" lorsque celle-ci ne doit pas être modifié et "<nom de la variable>= "valeur" lorsqu'elle peut l'être. Nous avons alors fait le choix de mettre tout les variables globales non modifiable sous la forme "#define <nom de la variable><valeur>". Cela permet d'avoir une lisibilité accrue, et économiser de l'espace mémoire. Des problèmes de compatibilité entre types pourraient apparaître, certes, mais nous n'utilisons les variables globales que comme types simples comme int ou str, tout en faisant attention à leur contexte d'utilisation, nous permettant ainsi d'éviter ces erreurs.

5 Conclusion

Ce projet nous a permis de mettre en pratique et de se familiariser avec les concepts vus en cours d'OS, tels que la gestion des processus, des signaux, la gestion de la mémoire partagée, et la communication inter-processus en C. Celui-ci, nous a appris à utiliser les outils de programmation en C comme sigaction, fork, les pipes nommés,... . Notons par ailleurs que tous nos projets de première année de bachelier se faisaient seul. Cette expérience est donc la première en groupe, impliquant certains avantages comme la répartition de tâches permettant à chaque membre d'avoir une charge de travail réduite, mais également d'autres aspects pas toujours positifs, comme la gestion d'équipe, qui peut parfois s'avérer laborieuse. Heureusement, cette gestion s'est faite sans accrocs dans notre groupe et nous a permis d'avoir un aperçu du travail de groupe dans le monde professionnel.