Rapport Projet système
Pour le xx Mai 2022

Chang Patrick, Du Vincent

May 15, 2022

1 Introduction

Le but du projet est d'implémenter les files de messages pour une communication entre processus tournant sur la même machine. Nous avons terminé le sujet initial avec l'extension 1 et 2. Nos files sont donc compactées et peuvent recevoir plus de messages que prévus.

Le travail a été souvent fait sur une même machine (ne pas prendre en compte le nom de l'auteur de chaque commit). Notre équipe étant composée de 2 personnes :

- **CHANG Patrick**, étudiant en M1 IMPAIRS à l'Université de Paris
- **DU Vincent**, étudiant en M1 IMPAIRS à l'Université de Paris

2 NOS PREMIERS CHOIX :

Nos premiers choix se portent tout d'habord sur le fichier `m_file.h`. Une structure `message_enreg` permet de représenter les messages. Cette structure contient un `int size` (taille), un `long type` (le type ex : pid), `char mtext[]` (le message en lui-même). La taille permet de pouvoir compacter la file et de naviguer dans chaque message.

La structure **ma_file** représente la file. Elle contient les éléments demandés dans le sujet. Nous n'avons pas utilisé de tableau circulaire ni l'element `first` car l'extension 1 ne le permet pas(compactage). Dans cette structure nous avons fait le choix d'utiliser `un mutex et 2 pthread_cond_t`. En effet, les `mutex` sont difficiles à vérifier(énormément de vérifications IF) mais moins facile de se tromper. Les 2 `pthread_cond_t` permettent de résoudre le problème du lecteur et de l'écrivain. **ma_file** contient une variable `size` précisant la taille des données écrites. Cela nous permettra de nous déplacer directement à la fin du message. Il faut alors faire attention à bien noter la bonne taille en octet.

La structure **MESSAGE** est bien séparée de la file pour permettre différents modes.

3 LES FONCTIONS :

`m_connexion` permet la connexion et la création d'une file de messages. Pour prendre différents nombres de paramètres nous avons utilisé une va_list. Lorsque nous créons une nouvelle file, nous donnons à l'objet en mémoire partagé une taille maximale possible par rapport au nombre maximum de messages stockables. C'est dans cette fonction que nous initialisons le `mutex` et les 2 conditions du `mutex`.

`m_deconnexion` fait un appel à la fonction unmmap sur la file du MESSAGE.

`m_destruction` détruit la file grâce à shm_unlink.

`m_envoi et m_reception` partent du même principe. Nous initialisons toutes les variables en dehors du `mutex_lock`. Nous vérifions si l'appel est bloquant. Si c'est le cas, les 2 variables conditions rentrent en jeu, sinon nous sortons directement de la fonction.

`m_envoi` place le nouveau message à la fin en ulisant la variable `size` de **ma_file**. `m_reception` supprime le message bien couvre le vide en compactant les autres messages. Une fois le bon emplacement trouvé les 2 fonctions changent directement les données pointées par le pointeur grace au mmap.

Nous utilisons le `mutex` lors de la création et de la suppression d'un `message_enreg`. Et lors du parcours des messages car un message peut être supprimé par un autre processus d'autant plus que les messages sont compactés (pas de tableau circulaire).

Le getter `m_nb` est entouré de lock car il fait partie de la file et c'est une variable pouvant être modifiée.

4 TESTS :

Le fichier `normalTest.c` contient différents tests sans autres processus. Il test la création, la destruction, la deconnexion d'une file, l'envoi et la reception d'un message.

Le fichier `forkTest.c` contient différents tests entre plusieurs processus en mode bloquant.

Le fichier `tests.c` contient les 2 précédents tests.

Le fichier `forkTest2.c` contient différents tests entre plusieurs processus en non bloquant.

Le fichier `forkTest3.c` contient un seul test bloquant.

Le fichier `ext2Test.c` contient un test pour l'extension 2.

5 EXTENSION :

Nous avons réalisé l'extension 1 en stockant pour chaque message leurs tailles et en donnant à la structure **ma_file** une variable `size`. Ces tailles permettent de compacter le tout mais il faut faire très attention à donner la bonne valeur. Nous n'avons donc pas implémenté de tableau circulaire.

Nous avons réalisé l'extension 2 en rajoutant un tableau contenant les pids et les signaux. En utilisant un **volatile sig_atomic_t** pour voir si un handler a été activé. Lorsqu'un message demandé arrive dans la file on envoie un signal. Il n'est pas pris en compte si un autre processus attend le même type de message.

6 LES PROBLEMES :

Le plus gros problème rencontré est le calcul de la bonne taille pour chaque message. En effet, lorsque nous manipulons les pointeurs une addition équivaut à prendre la taille de la structure et de la rajouter. Or ce que nous voulons c'est simplement rajouter une taille précise len. Pour cela nous convertissons le pointeur en un pointeur de `char` car celui-ci fait exactement 1 octet. Puis nous reconvertissons le pointeur vers son type de départ.