Manuel Technique

Par Samuel VICART

Table des matières

I. Introduction

Tout d'abord, commençons par énumérer les différentes composantes des programmes pour avoir une vue d'ensemble.

Le projet est décomposé en 3 parties, une bibliothèque proposant un système de file (Fifo), Un serveur exécutant des commandes et un client envoyant des commandes à exécuter au serveur. Chacune de ces trois parties est essentielle au bon fonctionnement du projet. Vous pouvez vous référer au manuel utilisateur si vous souhaitez en savoir plus sur la façon d'utiliser le serveur et le client.

Afin de communiquer, le client et le serveur utilisent 2 modes de communications distincts. Il s'agit de l'utilisation d'un tube nommé et d'une SHM (une mémoire partagée). Ces deux modes de communications sont détaillés dans de prochaines parties.

Durant tout le projet, de nombreuses solutions ont été développées pour permettre la communication entre le client et le serveur. Les plus importante d'entre elle vont être alors détaillée dans les parties à venir en commençant par la file (Fifo).

II. La file (Fifo)

La file est un des acteurs essentiels de ce projet. Elle prend ici la forme d'une bibliothèque possédant des structures et des commandes.

Il y a deux structures essentielles :

- La structure Commande (Elle contient la commande à envoyer au serveur ainsi que d'autres informations tel que le PID du processus appelant)

- La structure Fifo (Elle contient principalement un tableau de commande)

Pour ce qui est des commandes, nous retrouvons les commandes basiques d'une file, c'est-à-dire, défiler et enfiler. Elles ont pour but dans ce cas d'ajouter ou de retirer une commande du tableau dans la structure Fifo.

Nous trouvons également d'autres commandes utiles comme creer_fifo ou init_fifo. Ces commandes prennent en paramètre le nom de la file. En effet j'ai mis en place un système de nom pour les files. Cela permet de créer des noms de sémaphores celons le nom de de la file dans les commandes citées un peu plus hauts.

Pour finir, nous trouvons la commande « creer_commande ». Elle permet, comme son nom l'indique de créer une commande. Elle confectionne en fait une variable de type structure Commande, l'initialise et retourne un pointeur pointant sur cette nouvelle structure.

Pour que cette file fonctionne entre plusieurs processus il a été nécessaire d'implémenter un système de SHM ou mémoire partagée. C'est ce système que nous allons maintenant aborder.

III. La SHM (mémoire partagée)

Une SHM ou mémoire partagée est un espace dans lequel nous pouvons inscrire des informations. Ces informations peuvent alors par la suite être lues ou modifiées par d'autres processus. Pour ce projet j'ai mis en place une mémoire partagée contenant une structure Fifo (voir partie précédente). C'est donc une seule structure qui fait le lient entre les clients et le serveur (lanceur).

Cette SHM est créé dans le serveur est ouverte également en écriture dans les clients.

Bien entendu, comme toute mémoire partagé un système de sémaphores a été mis en place dans la bibliothèque de la file pour protéger son accès. Ce dernier permet donc la protection des données dans la SHM. Une lecture et Une écriture peuvent être effectuées à la fois. Ce système de sémaphores possède aussi l'avantage d'être bloquant. Cela permet alors d'effectuer une attente passive du serveur dans la fonction « defiler ». Ces Sémaphores sont également nommés et comme vu précédemment, ils le sont selon le nom de la file.

IV. Les threads

L'utilisation de threads pour le serveur a été primordial. Elle permet au serveur de pouvoir gérer plusieurs commandes à la fois.

Le fonctionnement de ces threads est simple. Nous avons un tableau de threads et un tableau de « drapeau » ou « flag ». Lorsqu'une commande arrive au serveur, le serveur cherche un thread disponible. Un thread est disponible si son drapeau est levé (il vaut 1). Si non son drapeau est baissé (il vaut 0).

S'il trouve un thread disponible il lui assigne la tache d'exécuter la commande qu'il vient de recevoir. Le thread baisse alors son drapeau et effectue la tache demandée.

Si tous les threads sont occupés il retourne alors au processus appelant une erreur via un signal (kill).

Dans les deux cas, une fois cette action effectuée, le processus principal cherche de nouveau à défiler une commande de la file. S'il n'y en a pas il attend alors passivement qu'une commande arrive.

Dès qu'un thread a fini l'exécution d'une commande et son envoie vers un tube, il lève son drapeau pour se rendre à nouveau disponible.

V. Les tubes nommés

Comme vu dans la parties précédentes, les tubes nommés servent de passerelles de retour pour les résultats des commandes exécutées par le serveur. Leur nom est demandé chaque exécution d'un nouveau client. Il est alors essentiel de changer de nom en fonction des clients pour le bon fonctionnement du programme. Un chiffre aléatoire ou le PID du processus appelant pourrait être un

bon moyen de rendre unique le nom de ce dernier tube mais il est ici laissé libre à l'utilisateur de choisir un nom qui lui convient.

Pour que l'on puisse facilement voir l'efficacité du programme j'ai décidé de mettre en place une écoute au niveau des clients. Concrètement, une fois que la commande a été enfilée dans la structure mise en SHM, le client attend de recevoir la réponse en ouvrant le tube nommé en écoute simple. Une fois que le client récupère les informations du tube il les affiches en utilisant a sortie standards.

VI. Conclusion

Finalement, de nombreuses solutions ont dû être mises en place pour mener à bien ce projet. Je vous invite alors à consulter les codes sources mis à disposition pour comprendre le fonctionnement précis des programmes et de leurs communications. Ces derniers sont annotés pour faciliter la lecture et la compréhension. Vous trouverez pour finir ci-dessous un schéma expliquant le fonctionnement des communications des programmes.