Rapport projet de systéme

Leroy Florent et Salles Théo

Décembre 2023

Table des matières

1 File synchronisée

La file synchronisée est construite comme une bibliothèque avec un fichier interface *file_sync.h* et un fichier implémentions. *file_sync.c*.

1.1 file_sync.h

Le module est composé de 4 fonctions. Une fonction de création de la file synchronisée, une qui libère les ressources allouées à la file et 2 autres fonctions pour ajouter ou retirer des éléments de cette file.

1.2 file_sync.c

Toutes les informations de la file telle que les sémaphores, le buffer de données ainsi que têtes de lecture et d'écriture sont conservées dans un segment de mémoire partagé qui est créer dans la fonction *create_file_sync* avant l'initialisation des sémaphores, des têtes de lectures et du buffer. Chaque action sur la file commence par l'ouverture du segment de mémoire partagée puis pour la fonction *destroy_file* on détruit toutes les sémaphores puis on libère le segment de mémoire partagée. Pour les fonctions *defiler* et *enfiler* c'est un simple problème consommateur/producteur avec les sémaphores.

2 Parseur

Le module parseur est une réutilisation du module *analyse* donner lors du tp 2 avec des tests rajoutée notamment sur les malloc.

3 Lanceur de commande

Le nom des tubes est définis par une normes choisis qui s'applique entre le lanceur et le client. Cette norme veut que le nom d'un tube soit une chaîne de caractère qui définit l'utilité du tube suivit du pid du client afin que chaque client possède des tube qui lui sont propres. Tout les tubes sont créer par le client.

3.1 Macroconstante

— TUBE_CLIENT_ : est le tube dans lequel le client écrit les informations pour le lanceur.
— TUBE_RES_CLIENT_ : est le tube dans lequel le resultat de la commande sera ecrit et envoyé au client.
— TUBE_ERR_CLIENT_ : est le tube dans lequel une possible erreur lors de l'exécution de la commande sera écrit et envoyé au client.
— BUF_SIZE : Taille maximale en nombre de caractère des commandes avec leurs options.
— CMD_SIZE : Taille maximale en nombre de du nom de la commande (taille déterminer à partir du nombre de caractère de la plus grande commande linux avec une marge de sécurité).
— PID_SIZE : Taille maximale en nombre de caractère du pid du client.

La structure *my_thread_args* contient tous les arguments à transmettre au thread.

3.2 main

Le principe du main est de créer un thread pour chaque demande d'exécution de commande. Nous commençons par faire du lanceur un deamon, pour cela nous faisons un fils dans lequel on le dissocie de la session. Puis on s'occupe de la gestion du signal pour terminer le lanceur. Ensuite tant que nous defilons un pid de client un créer un thread avec come argument le pid du client.

3.3 run

La fonction de lancement du thread créer un fils dans lequel on ouvre les 3 tubes lui permettant de communiquer avec le client. Puis lit les informations donner par le client sur le tube prévu à cette effet. Ensuite on redirige les 2 sortie standard vers les 2 tubes ouvert en écritures. On peut ensuite appeler la fonction *parseur_arg* du module *parseur* afin de récupérer la commande ainsi que chaque option de celle-ci. On fini par utiliser la fonction *execvp* afin d'exécuter la commande demander. Le pére lui ne fait que attendre son fils afin de libérer les ressources proprement.

4 Client

Le client récupère la commande à exécuter dans ses arguments lors de son appelle. On commence par générer le nom des tubes puis on créer ces tubes. Ensuite on envoie le pid du client dans la file afin que le lanceur puisse ouvrir les tubes. Maintenant on ouvre le tube de communication entre le client et le lanceur et on y écrit la commande à exécuter. On peut ensuite ouvrir les tubes de communication entre le lanceur et le client Puis en essaye de lire sur chaque tube et on l'affiche sur la sortie standard

5 Difficulté rencontré

Nous n'avons pas réussi à implementer l'utilisation des pipes lors du passage des commandes. Nous avons rencontrer des difficulté pour parser la commande dans le lanceur afin de pouvoir utiliser *execvp*. Dans le client nous avons eu des probléme quant à la récupération des informations issus des 2 tubes de réponses, fallait-il rediriger les sorties ou lire dans les tubes puis ecrire dans les sortir (solution qui fut choisis).