Projet système d'exploitation: serveur démon

Allognon Dieudonné Zanda-Vougo Maurel

2019-2020

Table des matières

1 Introduction

Dans ce projet nous essayons de réaliser un lanceur de commande, sous la forme d'un serveur qui répond aux requêtes émises par des clients. Le serveur gère a la fois un nombre de client limité par le nombre de thread que créé le démon. Lorsqu'aucun thread n'est disponible pour travailler le client reçoit un message pour le prévenir qu'il devra alors retenter une connexion plus tard.

2 Structures de données et fonctionnements de la file

2.1 Structures de données

Nous avons mis en œuvre le mécanisme de file pouvant accueillir n'importe quel type de donnée , accessible par le serveur et le client qui contiendra dans le cadre de ce projet des données de type requête, représentées par la structure request.

La structure **Fifo** est notre liste chainée elle contient les champs permettant une gestion optimale de la liste basée sur un algorithme producteur-consommateur . (image).

Elle contient les champs suivants :

- mutex est le mutex de notre algorithme permettant l'exclusion mutuelle,
- vide est une sémaphore qui permet de marquer la possibilité d'ajout d'un élément,
- plein est une sémaphore qui permet de marquer la possibilité de retrait d'un élément,
- tete est un entier représentant la position d'ajout dans le tampon,
- queue est un entier représentant la position de retrait dans le tampon,
- buffer est la variable tampon contenant les données

La structure request est la modélisation des données envoyées par le client au serveur permettant l'exécution et l'envoie de la réponse des commandes. Elle contient les champs :

- cmd qui est la commande a exécuté,
- args représentant les différents arguments de la commande s'il y en a,
- pipeName qui est le nom du tube servant à l'envoie du résultat,
- pid représente le pid du processus client initiateur de la commande.

2.2 création d'une file vide

La fonction create_fifo permet la création d'une file vide et prends en paramètre la taille. Elle crée également la shm et retourne un pointeur sur celle-ci.

2.3 Initialisation

La fonction prend en paramètre un objet de type Fifo et initialise les différents attributs de la structure a savoir les sémaphores (mutex, vide, plein) et les entiers (tete, queue)

2.4 Récupération de la file

La fonction **get_fifo** permet de récupérer un pointeur sur la file crée par create_fifo. Elle permet principalement au client d'avoir accès à la shm contenant la file et créée par le serveur

2.5 Enfiler

La fonction enfiler prends en paramètre tout type de donnée, un pointeur sur la file et la taille des informations et les enfile.

2.6 Défiler

La fonction défiler prends en paramètre la variable dans laquelle insérer l'élément a défiler, la file et la taille d'un élément .

3 Fonctionnement du serveur

Au lancement du serveur, la fonction create_fifo lancée ce qui permet la création de la shm contenant la file. Un tableau de threads dont la taille représente le nombre maximal de traitement simultané. Ces threads sont destinés a exécuté les commandes défilées. Le serveur associe également a chaque requête un entier qui informe sur le statut de la requête a l'aide d'une structure th_data.

.

Le serveur va défiler et tant que le nombre maximal de threads occupés simultanément n'est pas atteint, faire exécuter la commande par un thread libre. Si le nombre maximal est atteint, la requête est déclinée et le client reçoit SIGUSR1 .

L'exécution d'une commande par un thread implique , définir le thread comme occupé, détacher le thread, dupliquer ce dernier afin que le fils exécute la requête et envoie la réponse puis enfin, redéfinir le thread comme libre.

Le fils du thread en charge de l'exécution récupère le tube via une variable de type th_data contenant la requête, associe le tube a sa sortie standard et exécute la commande .

La fermeture du serveur est géré à l'aide de signaux. Lorsque nous tapon ctrl+C un signal est renvoyé dans notre serveur qui permet de mettre en action une fonction permettant de libérer la mémoire allouée par la shm.

4 Fonctionnement du client

Au lancement du client , celui-ci crée un tube, récupères la commande et éventuellement les arguments associés , les stockes dans une variable de type request avant de les enfiler dans la shm via la file.

Ensuite il reste en attente de lecture d'informations sur le tube , dès qu'il reçoit la réponse ou un signal d'interruption , les tube est supprimé et le programme arrêté.

5 Bilan

5.1 Difficulté

La gestion des accès au niveau de la zone de mémoire partagé n'était pas simple compte tenu de toute les situations critiques à éviter tel que l'interblocage, l'exclusion mutuel, la famine.

La gestion d'une file en librairie avec shm, stockage de tout type de données, et implémentation d'un algorithme permet la gestion efficace de la shm a représenté un challenge.

Nous avons eu des difficultés en ce qui concerne la méthode d'envoie des différentes informations au serveur , problème vite régler a l'aide des structures.

5.2 Efficacité

La mémoire étant une ressource précieuse, son utilisation doit respecter certaines exigences. Afin de bien gérer toutes nos allocations, nous avons pris en compte les différentes possibilités de terminaison des programmes serveur comme client et adapté les libérations de zone mémoire en conséquence. Le caractère détaché des threads est aussi essentiel en termes de gestion de la mémoire

5.3 Répartition du travail.

Dans un premier temps, les différents algorithmes, les structures et le mode de fonctionnement ont été travaillés et établis ensemble ce qui a permis d'aller vite en ce qui concerne l'implémentation.

En ce qui concerne la partie développement Mr ALLOGNON a pris en charge l'implémentation de la file, des structures de données et des phases d'initiation et de terminaison des programmes.

Mr ZANDA-VOUGO s'est donc occupé de l'implémentation de la gestion des threads , des requêtes, traitements et affichages.

Toutefois , chacun intervenait auprès de l'autre quand le besoin se faisait sentir.

5.4 Conclusion

Nous avons eu un réel plaisir à mettre en place ce serveur démon qui, sans le cacher demande encore beaucoup d'amélioration pour le rendre encore plus efficace. Dans ce projet, nous avons pu mettre en œuvre plusieurs notions vues en cours de programmation système et retravailler en TP comme les mutex, sémaphores, threads. Nous avons aussi pu manipuler certaines structures et fonctions POSIX.