Projet Systéme d'exploitation

Pierre Barthélemy, Mathieu Blondel

16 décembre 2018

Table des matières

1 Introduction

Réalisation d'un lanceur de commande en ligne de commande, le projet permétera de lancer des commandes à partir de plusieurs client en déffinissant des pipes comme entrée et sortie standard à ses commande. Les commandes sont éxécuter par un unique serveur. Le programme est développer en 3 partie, une file synchonisée, un serveur et un client.

file synchronisée

La liste d'attente synchronisée est réaliser à partie d'un espace de mémoire partagé, de deux sémaphore permétant l'accès à plusieurs processus, il ressoit les demandes des clients, se fait lire et vider par le serveur.

serveur

Le serveur va en récupérant la demande venant d'un client via la file synchonisée, éxécuter la commande voulus dans un nouveau procéssus avec les pipes fournis comme entrée et sortie standard.

client

Le client créer deux pipes pour l'entrée et la sortie standard, ajoute sa demande à la file d'attente, écoute et envois ses donnée sur les pipes.

2 File synchronisée

La file synchonisée utilise une structure de donnée pour permetre l'accée à ses fonctions :
La structure stocke le nom de l'espace de mémoire partagé et un descripteur de fichier. la liste des fonctions implémenter :

file_vide(void) Créer une structure correspondant à une file vide avec des éléments de taille fixe de taille size.
Renvois NULL en cas de dépassement de capacité mémoire ou si size <= 0
sinon renvois l'addresse d'un descripteur permétant d'y accéder.

file_ouvre(const char *name, int oflag, mode_t mod) Ouvre un espace mémoire éxistant et renvois un descripteur affin d'y accéder avec les autres fonctions de la librairie.
Renvois NULL en cas d'érreur.

file_ajout(const info *f, const void *ptr) . Ajoute l'éléments pointer par ptr à la fin de la file décrit pas f.
Si la file est pleine, tante de doubler la taille de la liste, en cas d'échec devient bloquant en attendant que la file se vide.
Renvois NULL si ptr == null ou en cas de dépassement de capacité mémoire, sinon renvois ptr.

file_retirer(info *f) Défile la file s, renvois nulle si la file et vide sinon renvois l'address d'une copie de l'élément qui était au début de la file, la libération de la mémoire lier à cette éléments est laisser à la discrétion de l'utisateur.
L'opération est bloquante tant en attendant de pouvoir dépiler un élément.

file_est_vide(const info *f) Renvois vrais ou faux si la file s est vide.

file_vider(info *f) libérer l'espace de mémoire partager et détruit les sémaphore, renvois -1 en cas d'érreur

La file est implémenter avec une liste flexible dans un espace mémoire partager, la liste peux voir sa taille doubler en cas de besoin pour permetre un plus grand nombre d'éléments.
Le créateur de la file doit définir une taille au éléments étant mis dans la file.
Pour permetre l'accée à plusieurs client, un algorithme producteur consomateur est implémenter avec 3 sémaphore stocker dans l'espace de mémoire partager.

3 Serveur

Le serveur lis la file synchonisée pour avoir les différentes commandes à éxécuter, quand une commande est ressus il créer un nouveau thread qui aura pour tache de s'en occuper.
Les demandes sont transmise à traver un structure regroupant diverses informations :

argv Liste des arguments de la commande, inclus la commande elle même

envp Arguments d'environement de la commande.

tube_in Nom du tube à relier à l'entrée standard de la commande.

tube_out Nom du tube à relier à la sortie standard de la commande.

La structure est modulable, la taille ainsi que le nombre d'arguments peuvent être changer via des constante écris dans le fichier lanceur.c Les tubes utiliser pour la commande sont ressus dans l'ordre :

tube_in : entrée standard permétant au client d'envoyer plus d'information à la commande.

tube_out : sortie standard permétant au client de recevoir les méssages de la commande.

La création des tubes est à la discrétion du client.
Dans le cas d'une commande transmise invalide, le serveur coupe le tube de retour sans rien écrire dedans. Le serveur ne se stope que dans le cas d'une érreur ou d'un signal d'arrét par l'utilisateur. Le serveur redéfinis le comportement des certains signaux.

SIGINT Libére la mémoire partager avant de stoper le programme.

SIGCHLD Wait l'enfant afin de gérer les zombies.

SIGCHLD Libére la mémoire partager avant de stoper le programme.

4 Client

Le client est le programme qui gére l'intéraction avec l'utilisateur, cela signifie qu'il passe par des entrées claviers. Le choix à était fais de prendre les demandes après l'éxécution via une similie d'interface graphique. Afin de pouvoir communiquer des pipes de communiquation à la commande demander par l'utilisateur le logiciel créer deux pipes à partir d'un nom type et de son identifiant afin de tanter d'obtenir des nouvelles pipes pas encore créer. La gestion de l'affichages se fais par un boucle lisant en permanence la pipe correspondante pendant que les entrées utilisateur son gérer par un deuxiéme thread avec des scanf.