Réalisation d’un lanceur de commandes

Réalisé par :

ARACHE Rayane

BENCHIKH Nassim

Noms des professeurs :

Yanick GUESNET
Djelloul ZIADI

Année Universitaire
2023/2024

Introduction

Ce Manuel technique décrit les solutions mises en œuvre pour réaliser les différents modes de communication entre le lanceur de commandes et ses clients. Le projet est divisé en trois parties distinctes : la bibliothèque de file synchronisée, le programme lanceur de commandes (démon), et le programme client. Le projet a été réalisé en collaboration à deux, avec une répartition des tâches organisée de manière stratégique.

1. Bibliothèque de File Synchronisée

1.1 Structure de la File Synchronisée (`fileSynchronisee`)

La file synchronisée est implémentée comme une structure en mémoire partagée (`shm_open` et `mmap`). Elle utilise des sémaphores (`sem_t`) pour assurer la synchronisation entre les différentes opérations.

```c
#define TAILLE_MAX_FILE 1000

struct fileSynchronisee {
    pid_t donnees[TAILLE_MAX_FILE];  // stock les pid des clients
    size_t taille;                   // la taille de la file entrée par l'utilisateur
    size_t tete;                     // la tête de la file
    size_t queue;                    // la queue de la file
    sem_t vide;                      // le nombre de places vides disponibles dans la file
    sem_t plein;                     // le nombre d'éléments présents dans la file
    sem_t mutex;                     // un sémaphore pour l'exclusion mutuelle
    bool estInitialisee;             /* l'état de la file, initialisée ou pas,
                                        doit être vrai pour pouvoir utiliser les autres commandes */
};
```

1.2 Opérations sur la File

```c
#define NOM_SHM "/file_shm"                 // le nom de la mémoire partagée
#define TAILLE_SHM sizeof(fileSynchronisee)  // la taille de la mémoire partagée

fileSynchronisee *initialiser_file(size_t taille_file);  // initialise la file en mémoire partagée avec la taille
void enfiler(fileSynchronisee *file, pid_t pid_client);  // enfile le pid d'un client
pid_t defiler(fileSynchronisee *file);                   // défile le pid du client et le retourne
void detruire_file(fileSynchronisee *file);              // supprime la file
bool file_vide(fileSynchronisee *file);                  // renvoie vrai si la file est vide, faux sinon
bool file_pleine(fileSynchronisee *file);                // renvoie vrai si la file est pleine, faux sinon
void ouvrir_file(fileSynchronisee **file);               // pour ouvrir une file déjà existante (par le client)

#endif
```

2. Programme Lanceur de Commandes (Démon)

2.1 Démarrage du Démon

Le démon démarre en créant une file synchronisée et en ouvrant un sémaphore pour la synchronisation avec les clients.

Le démon reste actif sans consommer les ressources, il n’est activé que par un client.

2.2 Activation du Démon par les Clients

Les clients activent le démon en signalant le sémaphore. Chaque client ajoute son PID à la file synchronisée.

2.3 Exécution des Commandes

Le démon utilise des threads dédiés pour traiter chaque commande. Les commandes sont extraites de la file, et un thread est créé pour les exécuter.

Le thread créer plusieurs processus (du même nombre que les commandes), chaque processus se charge de l’exécution d’une seule commande, les processus comuniquent entre eux avec des tube anonymes.

2.4 Gestion des Signaux

Le démon utilise un gestionnaire de signal (`SIGINT`) pour permettre une terminaison propre. Lors de la réception du signal, le démon ferme le sémaphore, supprime la file, et libère toutes les ressources.

3. Programme Client

3.1 Envoi de Commandes au Lanceur

Les clients envoient des commandes au lanceur en ajoutant leur PID à la file synchronisée, en créant des tubes nommés pour les entrées/sorties, et en signalant le sémaphore.

3.2 Exécution des Commandes par le Démon

Le démon récupère les commandes de la file, crée des threads pour les exécuter, et redirige les entrées/sorties vers les tubes nommés.

3.3 Récupération des Résultats

Les clients lisent les résultats des commandes depuis les tubes de sortie/erreur après avoir signalé le sémaphore.