# Projet :

Système d'exploitation

## Noms des membres de l'équipe :

- HAMEL Ferhat
- OUDIHAT Massinas

**Date :** December 31, 2024

# Contents

# 1 Introduction

L'objectif du projet est de mettre en place une architecture client/serveur dans laquelle le serveur se charge des tâches suivantes : réception des requêtes des clients pour calculer le produit de deux matrices d'entiers positifs. En outre, les matrices sont générées aléatoirement, selon les dimensions et la borne supérieure que le client a envoyées. Les sections suivantes exposent les solutions réalisées et les décisions techniques faites pour se conformer aux spécifications requises.

# 2 Objectifs

- Mettre en place une communication entre le client et le serveur via des tubes nommés.
- Implémenter une architecture robuste capable de gérer plusieurs clients en parallèle.
- Réaliser le produit matriciel en utilisant des threads.

# 3 Structure du Projet

Le projet est organisé comme suit :

```
|- ./ (répertoire racine du projet)
|   |- src/ (répertoire des modules du projet)
|   |   |- matrix/ (module matrix)
|   |   |   |- matrix.c
|   |   |   \- matrix.h
|   |   |- segment/ (module segment)
|   |   |   |- segment.c
|   |   |   \- segment.h
|   |   |- utils/ (module utils)
|   |   |   |- utils.c
|   |   |   \- utils.h
|   |   |- workera/ (module workera)
|   |   |   |- workera.c
|   |   |   \- workera.h
|   |   \- workerb/ (module workerb)
|   |       |- workerb.c
|   |       \- workerb.h
|   |- obj/ (répertoire des fichiers objet et des fichiers de dépendance)
|   |- main.c (fichier contenant la fonction main du serveur)
|   |- client.c (fichier contenant la fonction main d'un programme client)
|   |- Rapport_Projet_SE.pdf
|   \- makefile
```

# 4 Manuel Utilisateur

## 4.1 Étapes

### 4.1.1 Compilation

Exécutez la commande `make` dans le répertoire racine du projet.

### 4.1.2 Exécution

- Démarrez le serveur : `./server`
- Lancez le client dans un autre terminal : `./client`

# 5 Manuel Technique

## 5.1 Modules développés pour le projet

Les fonctions des modules suivants ne garantissent aucune vérification sur la conformité de leurs paramètres, c'est à l'utilisateur de s'assurer de la consistance des appels à ces fonctions.

### 5.1.1 Matrix

`matrix` est un module que nous avons développé pour gérer exclusivement les matrices A, B et C du projet qui ont pour éléments des entiers.

Il inclut entre autres :

- Un type pour les matrices.
- Une fonction d'initialisation d'une matrice dans un espace mémoire.
- Une fonction `matrix_get_cell` qui renvoie un pointeur vers une cellule de la matrice représentée par un numéro de ligne et un numéro de colonne.
- Une fonction `matrix_write` qui copie les éléments de la matrice dans un emplacement mémoire en suivant d'abord l'ordre selon le numéro de ligne et ensuite l'ordre selon le numéro de colonne.
- Des fonctions de consultation des paramétrages comme le nombre de lignes, le nombre de colonnes, la taille en octets nécessaire pour stocker une matrice et la taille en octets occupée par les éléments d'une matrice donnée.
- Une fonction de verrouillage et une fonction de déverrouillage pour synchroniser l'accès à la matrice entre des threads ou des processus.

### 5.1.2 Segment

`segment` est un module que nous avons développé pour gérer exclusivement le segment de mémoire partagé nécessaire au projet.

Il inclut entre autres :

- Un type pour les segments.
- Une fonction d'initialisation d'un segment de mémoire partagé.
- Une fonction qui écrit les éléments des matrices A, B et C dans un fichier déjà ouvert.
- Des fonctions de consultation des paramétrages :
  - Pointeur vers l'une des matrices A, B ou C.
  - La taille en octets occupée par le segment.
- Des fonctions utilitaires sur les opérations de verrouillage et déverrouillage des matrices A, B et C.

### 5.1.3 Workera et Workerb

`workera` et `workerb` sont deux modules offrant les deux fonctions `worker_a` et `worker_b` qui servent à encapsuler une partie du travail de `workera` et `workerb` décrits dans le projet.

# 6 Stratégie de développement et fonctionnement

## 6.1 Développement des sources

Le développement des sources du projet est organisé avec le système de gestion de versions `git`. Le dépôt sur GitHub est accessible par le lien suivant :
`https://github.com/hamelfer/projet_se`.

### 6.1.1 Server

Au lancement du programme, il ouvre en lecture le tube nommé `server_requests_pipe` qui est le tube sur lequel les demandes de clients sont transmises. À la réception d'une requête complète, le programme lance un processus de type `workera` (voir 6.1.2), puis se met a attendre qu'un des processus cree se termine et enfin il attend une autre requette.

### 6.1.2 Processus de type Workera

À l'aide de `fork`, le processus fraîchement créé aura accès aux paramètres de la requête et procèdera à la préparation de la réponse à envoyer au client en appelant la fonction `worker_a` avec les paramètres préétablis. L'appel à cette fonction génère :

- La création d'un segment de mémoire partagé et sa projection dans la mémoire.
- Le lancement d'un processus de type `workerb` (voir 6.1.3).

- L'initialisation de la matrice A (voir 6.1.5).
- La transmission du résultat au client via un tube nommé créé par le client (voir 6.1.7).

### 6.1.3 Processus de type Workerb

Le processus récemment créé aura pour tâche d'appeler la fonction `worker_b` qui initialise la matrice B (voir 6.1.5) et calcule la matrice C (voir 6.1.6).

### 6.1.4 Synchronisation

Les deux processus créés suite à la réception de la requête se synchronisent grâce aux fonctionnalités de verrouillage et déverrouillage des matrices du module `matrix` et à l'appel système `wait`.

### 6.1.5 Initialisation des Matrices

L'initialisation des matrices se fait avec des entiers aléatoirement générés basés sur la fonction `rand`.

### 6.1.6 Calcul de la Matrice C

La matrice C est le produit des deux matrices A et B. Le calcul se fait par $(n \times p)$ threads.

### 6.1.7 Normes pour les Tubes Nommés

Les tubes étant nommés, la norme a dû être imposée :

- Le nom du tube des requêtes doit être `server_requests_pipe`.
- Le nom d'un tube de réponse doit être la représentation du `pid` du client en chaîne de caractères. Par exemple, la réponse de la requête du client de `pid` 123 sera transmise sur le tube nommé `123`.

# 7 Améliorations Possibles

- Ajout d'un fichier de configuration pour personnaliser les paramètres du serveur.
- L'utilisation du `timeout` pour l'attente de la réponse.
- La transformation du serveur en démon.

# 8 Conclusion

Ce projet démontre une mise en œuvre réussie d'une architecture modulaire pour un système client-serveur. La structure claire facilite la maintenance et l'extension du code pour inclure de nouvelles fonctionnalités.