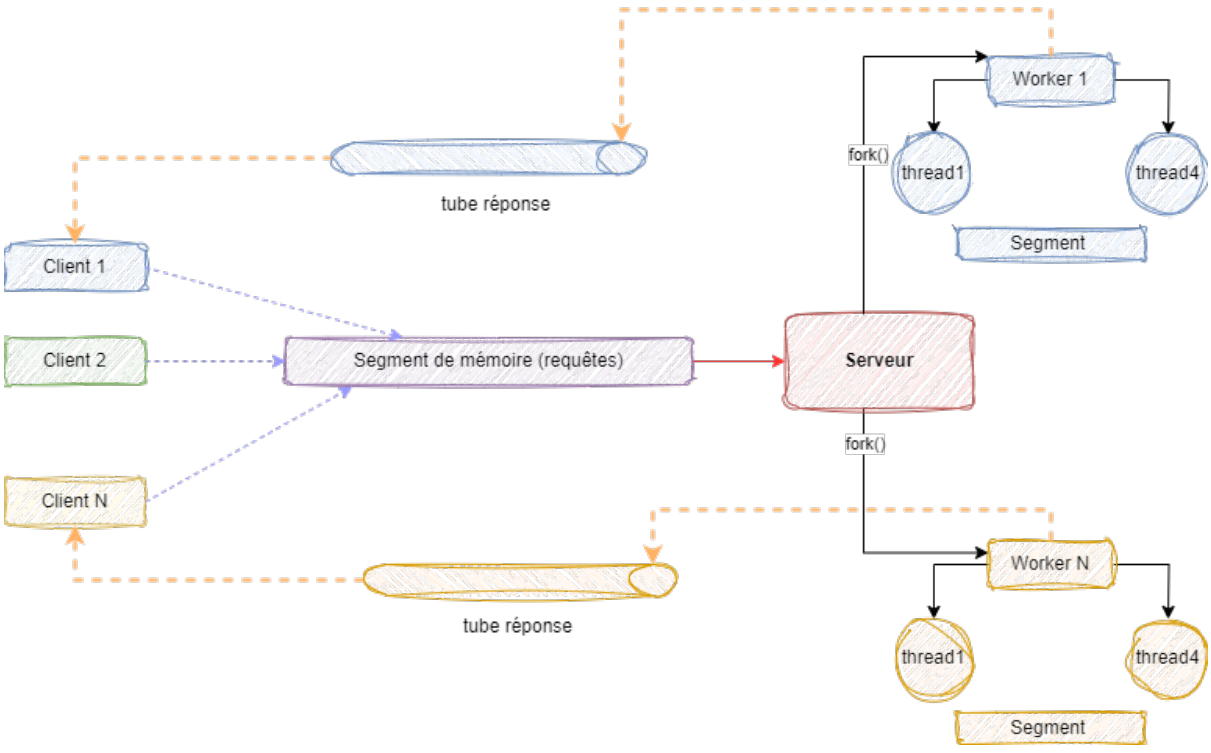
# Synthèse

## 1 Sujet

Le projet consiste à concevoir et implémenter une architecture client/serveur permettant à des clients de soumettre des requêtes de filtrage d'images. Le serveur délègue le traitement à des processus workers utilisant la mémoire partagée et le multithreading, et retourne le résultat directement au client via une FIFO dédiée.

**Architecture :** L'architecture se compose d'un client, d'un serveur maître et de workers. Le client soumet des requêtes et reçoit l'image traitée via une FIFO. Le serveur orchestre les workers, qui appliquent les filtres en parallèle avec des threads coopératifs. Les segments mémoire sont utilisés uniquement pour le partage entre threads d'un même worker.




**Protocole de communication :** Le client envoie une structure contenant son PID, le chemin de l'image à traiter, le type de filtre à appliquer,et les paramètres éventuels du filtrage. Le worker écrit l'image traitée directement dans la fifo /tmp/fifo_rep_<pid client>, que le client lit pour reconstruire l'image finale.

```
struct filter_request {
        pid_t pid;
        char chemin[256];
        int filtre;
        int parametres[5];
};
```

**Déroulement :** Le client crée sa FIFO de réponse (/tmp/fifo_rep_<son_pid>) et dépose la requête dans le segment partagé (requêtes). Le serveur surveille ce segment, crée un worker pour chaque requête et lui transmet les détails du traitement. Le worker charge l'image, divise le travail entre les threads (4 à 8 threads) et applique le filtre en parallèle. Enfin, le worker écrit l'image traitée dans la FIFO (/tmp/-fifo_rep_<pid_client>), et le client la lit pour obtenir le résultat.

**Threads :** Chaque thread du worker travaille sur un segment de l'image (**BMP**). La division se fait par bandes horizontales avec synchronisation éventuelle (sémaphores ou pthread_join() si les threads sont joinables). La gestion des pixels aux bordures est nécessaire uniquement pour les filtres basés sur les voisins (ex. Sobel, flou), mais pas pour les filtres indépendants par pixel (gris, négatif, luminosité/contraste). Dans ce projet, **le filtre "niveau de gris" est obligatoire**. Les autres filtres (négatif, luminosité/contraste, etc.) et le support d'autres formats d'images sont en bonus. Rappelons le principe du filtre niveau de gris : Une image couleur utilise généralement 3 composantes (Rouge, Vert, Bleu - RVB). Le filtre niveau de gris combine ces trois canaux pour obtenir une seule valeur d'intensité lumineuse. La méthode la plus courante utilise une moyenne pondérée qui tient compte de la sensibilité de l'oeil humain aux différentes couleurs :

$$\text{Gris} = 0.299 \times \text{Rouge} + 0.587 \times \text{Vert} + 0.114 \times \text{Bleu}$$

Par la suite chaque pixel de l'image devient un pixel gris dont l'intensité varie du noir (0) au blanc (255), créant une image monochrome.

```
struct thread_workspace {
        int thread_id;
        struct segment_image *shm;
        int ligne_debut;
        int ligne_fin;
        int filtre;
        // ...
};
```

## 2 Travail à réaliser

En plus des codes sources correctement documentés et d'un makefile respectant les options spécifiées en début de semestre, vous rendrez :

- un petit manuel utilisateur explicitant comment utiliser votre application ;
- un manuel technique décrivant les solutions que vous avez été amené.e.s à développer pour réaliser les différents modes de communication entre le serveur et ses clients.

**Contraintes techniques :** Communication via FIFO pour le retour et mémoire partagée pour threads, multithreading coopératif (4-8 threads par worker), gestion des ressources, synchronisation et erreurs (fichiers, mémoire, timeout).
Votre programme doit traiter les images au format **BMP (bitmap)**. La taille d'une image ne doit pas dépasser 100M.
Le filtre **"niveau de gris"** doit être implémenté obligatoirement.

**Bonus :** Implémentation d'autres filtres (négatif, luminosité/contraste, etc.) et support d'autres formats d'images (PNG, JPEG, etc.).

**Critères d'évaluation :** Fonctionnalité des filtres (25%), coopération des threads (20%), gestion des FIFOs et synchronisation (20%), robustesse (15%), performance et parallélisme (10%), documentation et qualité du code (10%).

Le projet peut être réalisé seul ou en binôme. La date limite de rendu est fixée au **mercredi 31 décembre 2025 23h59**. Une soutenance sera organisée en début d'année 2026.