Simulation d’un Restaurant — Producteurs / Consommateurs

1. Présentation du projet

1.1 Nom du projet

Simulation d’un Restaurant — Producteurs / Consommateurs

1.2 Objectif

Simuler le fonctionnement d’un restaurant en environnement concurrent.
Les serveurs produisent des commandes, les cuisiniers les consomment.

La synchronisation repose sur des **threads**, **mutex**, **sémaphores** et un **buffer circulaire**.

1.3 Fonctionnalités principales

- Saisie interactive des commandes
- File d’attente via buffer circulaire
- Préparation parallèle par plusieurs cuisiniers
- Synchronisation avancée (mutex + sémaphores)
- Logs HTML lisibles dans un navigateur
- Affichage en temps réel

2. Architecture technique

2.1 Organisation du projet

Code
commande.h/.c : Structure Commande
interaction.h/.c : Saisie utilisateur
file.h/.c : Buffer circulaire
serveur.h/.c : Threads serveurs (producteurs)
cuisinier.h/.c : Threads cuisiniers (consommateurs)
log.h/.c : Logs HTML
globals.h/.c : Mutex, sémaphores, compteurs
main.c : Initialisation + orchestration

2.2 Structures et fonctions principales

Structure Commande

- id unique
- plat
- temps de préparation
- état (PRISE, EN_PREPARATION, TERMINEE)

Fonctions essentielles

- `saisir_commande()` : lit le plat et calcule le temps
- `ajouter_commande()` / `retirer_commande()` : gestion du buffer
- `serveur()` : producteur
- `cuisinier()` : consommateur
- `write_log()` : écriture dans logs.html

2.3 Gestion des threads et synchronisation

Threads

- 2 serveurs
- N cuisiniers (`NB_CUISINIERS`)

Mutex

- `mutex_file` : protège la file
- `mutex_compteur` : protège les compteurs
- `mutex_affichage` : évite les affichages simultanés
- `mutex_saisie` : empêche les saisies concurrentes

Sémaphores

- `places_libres` : capacité restante
- `commandes_disponibles` : commandes prêtes
- `cuisiniers_disponibles` : cuisiniers libres

Modèle producteur–consommateur

- Serveurs → produisent
- Cuisiniers → consomment
- Sémaphores → garantissent la cohérence

3. Instructions d’utilisation

3.1 Compilation

Code
gcc -pthread *.c -o restaurant

3.2 Exécution

Code
./restaurant

3.3 Dépendances

- GCC avec support POSIX threads
- pthread, semaphore, unistd, time

3.4 Fichiers générés

- `restaurant` : exécutable
- `logs.html` : journal des actions

4. Tests effectués

Test 1 — Saisie

Entrées : pizza, grec, burger → temps corrects, commandes ajoutées.

Test 2 — File pleine

Serveurs rapides, cuisiniers lents → attente sur `places_libres`.

Test 3 — Tous cuisiniers occupés

`sem_trywait()` → message “mise en file d’attente” + log.

Test 4 — Préparation

Décompte affiché → comportement cohérent.