Labo 03 PCO

Maxime Regenass – Nathan Füllemann

Introduction

Ce projet de laboratoire consiste à simuler une crise sanitaire dans une petite ville en mettant en place un système de gestion d'accès concurrents avec plusieurs entités collaboratives (ambulances, hôpitaux, cliniques, et fournisseurs). Chaque entité est représentée par un thread, et des mécanismes de synchronisation sont utilisés pour assurer la sécurité des accès partagés.

Objectifs

- Simuler les interactions et la logistique entre les entités de santé.
- Protéger les accès concurrents aux ressources partagées (stocks, fonds) en utilisant des mutex.
- Gérer la fin de la simulation de manière propre en arrêtant tous les threads et en vérifiant la cohérence des données.

1. Description de l'Implémentation

1.1 Structure des Classes

Le projet est structuré autour des classes principales suivantes :

- **Ambulance** : Transporte les patients vers les hôpitaux.
- **Hospital** : Accueille et soigne les patients. Gère des lits limités et libère les patients guéris.
- **Clinic** : Traite les patients en utilisant des ressources médicales spécifiques.
- **Supplier** : Fournit les ressources médicales nécessaires aux autres entités.

- **Seller (Classe de Base)** : Fournit une interface commune pour les transactions et les méthodes de base utilisées par les entités.

Chaque classe contient une routine run() exécutée dans un thread distinct, simulant l'activité continue de l'entité (transfert de patients, commande de ressources, etc.).

1.2 Gestion des Accès Concurrents

Pour éviter les conflits d'accès sur les ressources partagées (stocks et fonds), des mutex ont été utilisés. Les sections critiques incluent :

- **Les transactions financières** : Lorsque les fonds d'une entité sont modifiés, par exemple lors d'un transfert de patient.
- **Les stocks** : Lorsqu'une entité ajuste ses stocks de ressources ou de patients.

1.3 Gestion des Transactions

Chaque entité dispose de méthodes send et request pour gérer les échanges de ressources entre elles. Ces méthodes vérifient que l'entité peut bien transférer ou recevoir la ressource demandée (ex. vérification des fonds, disponibilité du stock).

1.4 Arrêt des Threads et Fin de la Simulation

Pour gérer la fin de la simulation, nous avons implémenté la fonction Utils::endService() qui :

1. Demande à chaque thread de s'arrêter en appelant stopRequested().
2. Attache (join) chaque thread principal pour garantir qu'il termine proprement.
3. Calcule un récapitulatif des fonds et stocks pour vérifier la cohérence finale des données.

2. Étapes de Développement

Étape 1 : Création des Structures de Base

- Nous avons implémenté les classes principales en respectant l'architecture fournie, en s'assurant que chaque classe dispose de ses attributs spécifiques (stocks, fonds, etc.).
- Mise en place des mutex pour la protection des sections critiques.

Étape 2 : Implémentation des Interactions et des Méthodes

- Développement des méthodes send et request pour gérer les échanges de patients et de ressources entre les entités.
- Intégration des coûts et des salaires dans les transactions pour simuler un système de gestion financière.

Étape 3 : Gestion de la Concurrence et de la Cohérence

- Utilisation de Google Test pour vérifier les interactions, comme le transfert de patients et la commande de ressources.
- Tests de résistance pour simuler des accès concurrents à un même stock et s'assurer que les mutex fonctionnent correctement.

Étape 4 : Gestion de la Fin de Simulation

- Implémentation de Utils::endService() pour une terminaison ordonnée.
- Calcul et affichage des totaux de fonds et de stocks pour vérifier l'intégrité de la simulation.

3. Vérification et Tests

Pour vérifier le bon fonctionnement et la cohérence des données dans un environnement concurrentiel, nous avons utilisé des tests unitaires Google Test :

1. **Test des transferts de patients** : Vérification que le nombre de patients diminue bien dans l'ambulance et augmente dans l'hôpital après chaque transfert.
2. **Test des commandes de ressources** : Test des méthodes orderResources de la Clinic et request du Supplier pour s'assurer que les ressources sont correctement achetées.
3. **Test de cohérence des stocks et fonds** : Après plusieurs cycles de transactions, vérification que les fonds et stocks sont cohérents, sans perte ni création inattendue de ressources.

Conclusion

Ce projet a permis de mettre en pratique la gestion d'accès concurrents dans un système simulé complexe. Les tests unitaires ont permis de vérifier la cohérence des transactions dans un environnement multi-threadé, offrant une solution fiable à la gestion de crise simulée.