Laboratoire 3 - Mutex in the medical domain

Calum Quinn - Urs Behrmann

Table des matières

Introduction au problème

Le but primaire de ce laboratoire est d'implémenter la gestion de multiples threads concernant la concurrence et surtout l'accès concurrent à des variables partagées.

Pour celui-ci nous avons dû tout d'abord implémenter les opérations d'échanges entre les divers acteurs du labo (ambulances, clinics, hôpitaux et fournisseurs).

Une fois ceci effectué, nous avons pu nous pencher sur les aspects de concurrence et donc la protection des zones critiques lors des modifications des variables partagées.

Choix d'implémentation

Libération des patients soignés

Pour l'implémentation des jours de repos obligatoire, nous avons décider d'utiliser un vecteur comme attribut de la classe.

Ceci permet de facilement savoir combien de personnes ont passés combien de temps au repos après avoir été soigné et donc de les libérer dès les 5 jours effectués.

Gestion de la concurrence

Ce projet comportait diverses instances de concurrence. Les situations en question se produisent lors des accès concurrents à la même ressource d'un même établissement. Par exemple, deux hôpitaux différents peuvent simultanément demander un patient soigné dans la même clinique.

Il a donc fallu gérer les accès concurrents dans les fonctions `request()` des établissements (hôpital, clinique, fournisseur), la fonction `send()` de l'hôpital, ainsi que les accès aux variables propres des établissements.

Les mutex utilisés dans une classe garantissent qu'un seul thread à la fois peut accéder aux variables de cette classe. Même la classe à laquelle les variables appartiennent doit attendre que les autres threads aient terminé leurs requêtes pour accéder à ces informations.

La seule exception est l'ambulance, qui n'a pas besoin de mutex. Elle se contente d'envoyer des requêtes par elle-même et n'a pas besoin de prendre de décisions en fonction des autres threads, car aucune requête ne lui est adressée.

Arrêt des threads lors de la fermeture

Pour arrêter proprement les threads lors de la fermeture de la fenêtre, nous envoyons une demande d'arrêt à chaque thread (`requestStop()`).

Pour que la demande soit exécutée, il faut introduire le contrôle de la variable comme condition de continuation de chaque thread.

C'est-à-dire que si la variable `stopRequested` passe à `true` le thread comprend qu'il doit se terminer.

Patients malades créés par les fournisseurs

Il y avait un problème où des patients malades étaient créés par les fournisseurs, c'est pourquoi il a été nécessaire d'ajouter une condition lors de la création de nouveaux objets fournis.

```
std::find(this->resourcesSupplied.begin(),  this->resourcesSupplied.end(),  resourceSupplied)  !=
this->resourcesSupplied.end()
```

Chaque fournisseur possède une liste d'objets qu'il peut créer. On vérifie si l'objet créé fait partie de cette liste. Si c'est le cas, l'objet est ajouté à la réserve du fournisseur ; sinon, le programme poursuit son exécution.

Tests effectués

Calculs d'argent

Nous sommes passé à travers toutes les modifications d'argent pour confirmer que celles-ci se compensent à chaque fois afin de conserver le total.

E.G.

- Transfert sortant compense transfert entrant
- Achat de matériel compense la vente du même matériel

Ceci ne s'applique pas aux salaires car elles sortent définitivement du système. Il a tout de même fallu contrôler que celles-ci soient bien payées au moment nécessaire.

Calculs de patients

Nous sommes également passés à travers toutes les modifications du nombre de patients pour faire de même qu'avec l'argent pour conserver le nombre total.

Concurrence

Une fois que les éléments de gestion de la concurrence, par exemple les mutex, étaient mis en place, nous avons pu tester son bon fonctionnement en contrôlant justement les deux éléments précédents. Le bon calcul de l'argent et des patients repose en grande partie sur la gestion correct des variables partagées qui décompte les ressources.

Puisque le calcul est à chaque fois juste, il est très peu probable que la concurrence des threads pose des problèmes.