Rapport sur les différents design pattern utilisés

Dans les projets de programmation concurrente en C++, plusieurs design patterns peuvent être particulièrement utiles pour gérer la complexité liée à la concurrence. Voici quelques-uns des types de design patterns que nous avons eu à utiliser durant le projet :

1. **Patrons de création**

- Thread Pool : Permet de gérer un ensemble de threads qui peuvent être réutilisés pour exécuter des tâches, évitant ainsi le coût d'une création de thread à chaque fois.

2. **Patrons structurels**

- Proxy : Utilisé pour contrôler l'accès à des objets, permettant de gérer les appels de méthodes de manière asynchrone ou de synchroniser l'accès à des ressources partagées.

- Facade : Fournit une interface simplifiée pour interagir avec des sous-systèmes complexes, comme les bibliothèques de gestion de threads.

3. **Patrons comportementaux**

- Observer : Permet de mettre en place un mécanisme de notification entre objets, ce qui est utile pour les systèmes réactifs où les changements d'état doivent être communiqués à plusieurs parties.

- Command : Encapsule des requêtes en tant qu'objets, permettant de les exécuter de manière asynchrone, d'annuler des commandes ou de les mettre en file d'attente. Mediator : Facilite la communication entre objets en centralisant les interactions, ce qui peut aider à gérer les dépendances dans un environnement concurrent.

4. **Patrons de synchronisation**

- Monitor : Encapsule l'accès à une ressource partagée en utilisant des mécanismes de verrouillage pour protéger les sections critiques.

- Read/Write Lock : Permet de gérer l'accès concurrent à une ressource, en permettant plusieurs lecteurs ou un seul écrivain, optimisant ainsi les performances lorsque la lecture est plus fréquente que l'écriture.

5. **Patrons de coordination**

- Future/Promise : Permet de gérer des résultats d'opérations asynchrones, où un objet `Future` représente une valeur qui sera disponible à un moment donné.

- Barrier : Synchronise plusieurs threads à un point donné, en les bloquant jusqu'à ce que tous atteignent ce point.

Ces design patterns aident à structurer le code concurrent de manière à améliorer la lisibilité, la maintenabilité et la performance des applications C++. En les intégrant judicieusement, nous pourrions mieux gérer la complexité inhérente à la programmation concurrente, tout en minimisant les risques d'erreurs comme les conditions de course et les interblocages.