CAS #8 : Optimiseur de performance

8INF957 : Programmation objet avancée
Pondération : 10%
Consignes :

- Travail en équipes de 3
- Remise par Moodle
  - Un seul projet, compilé et fonctionnel
  - Document de preuve de fonctionnement
  - Le support de présentation
  - Le rapport

La mise en situation

Un système effectue le traitement de requête provenant de divers fournisseurs (F). Chaque fournisseur effectue ses envois de manière asynchrone et chaque requête est insérée et entreposée dans un transformateur (T)

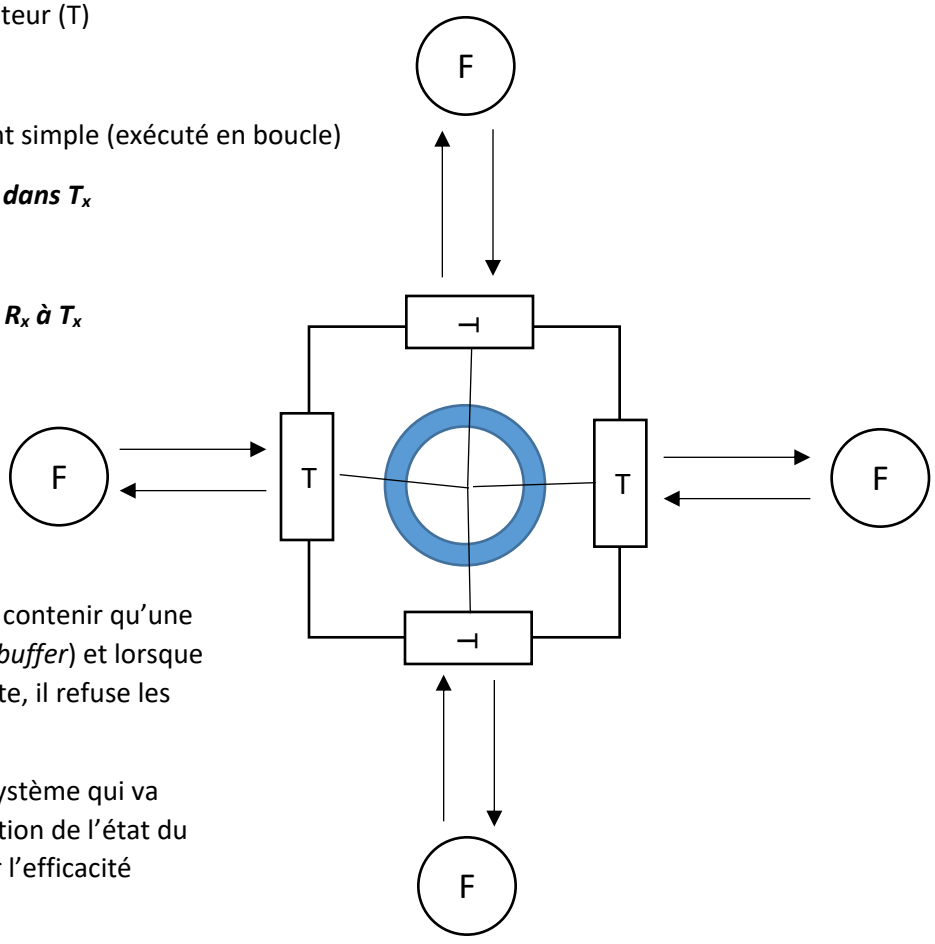
Le système a un fonctionnement simple (exécuté en boucle)

Récupérer une requête dans T_x

Traiter T_x

Retourner une réponse R_x à T_x

Passer à T_{x+1}



Le problème est que T_x ne peut contenir qu'une certaine quantité de requêtes (*buffer*) et lorsque la capacité maximum est atteinte, il refuse les nouvelles requêtes.

Votre mandat est de créer un système qui va optimiser le traitement en fonction de l'état du système et donc d'en améliorer l'efficacité

Détail

Le fournisseur « F »

- Chaque fournisseur est un Thread
- Il exécute les fonctions suivantes
  - Générer des requêtes (temps de traitement variable – Sleep avec random)
  - Recevoir des réponses/échecs
  - Journaliser les évènements (ex : si lancé en mode Debug, écrit dans la console)

Le transformateur « T »

- T_x est associé à F_x
- Un transformateur peut contenir une capacité limité de requêtes en mémoire
  - Par exemple, avec une capacité de 10
  - VIDE (0), OK (3), DANGER (7) ET MAX (10)
- Toute requête dépassant MAX est automatiquement rejetée et retourne un échec à F_x
- Chaque T est un objet qui peut réaliser les fonctions suivantes
  - Recevoir une requête
  - Retourner une réponse/échec

Le système « S »

- Composé d'une liste de transformateurs
- Le système démarre toujours avec la stratégie de traitement $S_{défaut}$
- Après un certain nombre de traitements (à déterminer), le système tombe en maintenance et affiche les statistiques de performance (voir page suivante)

Les solutions

Projet #1 : Créer un projet qui implémente la stratégie par défaut et afficher les statistiques (exécuter plusieurs fois pour avoir une moyenne significative)

Projet #2 : Créer un projet qui implémente des stratégies optimisant la performance du système et afficher les statistiques (exécuter plusieurs fois pour avoir une moyenne significative)

Le calcul des statistiques de performance

Voici les coûts

- Pour chaque traitement par le système
  - $S_{Défaut}$: 1
  - $D_{Débalancement}$: 2
  - $S_{Surcharge}$: 3
- Pour chaque rejet : 25

Donc, il faut calculer le cout de traitement pur toutes les opérations du système entre le moment de démarrage et l'arrêt de maintenance

Les stratégies

- Stratégie par défaut ($S_{Défaut}$) : Le système tourne en rond. Voir la mise en situation
- Stratégie par débalancement ($D_{Débalancement}$) : Lorsqu'un T_x est plus plein que les autres T alors on rétabli l'équilibre puis on revient à $S_{Défaut}$
- Stratégie de surcharge ($S_{Surcharge)}$: Lorsque qu'un T_x est dangeureusement plein, on le vide jusqu'à OK puis on revient à $S_{Défaut}$
- Tout autre stragégie qui vous semble adéquate
- Vous devez trouver un mécanisme de gestion des stratégies le plus objet possible! (*une machine d'état?*)

Contraintes

- Le système doit être objet
- L'objectif est l'optimisation du système et de pouvoir comparer avantageusement avec le mode par défaut
- Le temps d'exécution n'est pas l'unité de mesure mais bien le coût de traitement
- Pour le développement du prototype
  - Balancer la quantité de Tx et la vitesse de traitement des requêtes pour avoir des résultats significatifs
  - Il est possible que l'affichage ait un impact important sur l'exécution de votre programme (avec des « sleeps » assez bas) alors il serait préférable de limiter les affichages au maximum (utiliser un mode « debug »)
  - Attention aux méthodes « synchronized » (oui et non, pas n'importe où)