PROCESSUS CONCURRENTS ET PARALLÉLISME — IFT630 — Hiver 2024

*Devoir à remettre au plus tard le : 19 mars 2024 à 23:59*

# Projet 2

## § Communication Inter-Processus §

**Instructions de compilation et de remise**

- Inscrire ton cip dans le debut du nom du fichier zip à remettre.
- Utiliser le dossier projet2 dans l'outil tunin pour remettre votre devoir avant la date de remise.

## INSTRUCTIONS

`Travail à réaliser uniquement en langage Java.` **Chaque problème vaut 5 points**.

## Société de transport de Sherbrooke

La **Société de transport de Sherbrooke** (STS) est l'opérateur des services de transport en commun qui couvre le territoire de Sherbrooke au Québec. Selon le rapport annuel de 2022, la STS a transporté plus de 4 millions de passagers en ayant parcouru 8 071 459 km. En 2022, l'organisme parapublic a assuré le service de transport urbain sur 57 lignes, dont 18 sont desservies par des autobus urbains, 11 par des minibus, 3 lignes en microbus, 2 lignes en taxi bus, ainsi que 4 linges directes, 7 lignes express, 9 trajets spéciaux et 3 lignes de transport à la demande (TAD). De plus, elle a transporté 237 012 passagers et parcouru 786 258 km au transport adapté.[1].

## Communication Inter-processus entre les entités de la STS

La communication entre les entités au sein de la STS peut se faire à l'aide de divers mécanismes d'inter-processus en fonction des besoins spécifiques du système.

**(1) Mémoire partagée** les entités s'exécutent de manière asynchrone et utilisent la mémoire partagée pour permettre un échange rapide de données entre les processus (thread).

**(2) Bus de messages** La mise en place d'un bus de messages interne au sein du système de la STS permettrait aux entités de publier et de souscrire à des messages, facilitant ainsi la communication asynchrone entre les différents composants du système.

**(3) Les signaux** Chaque entité peut envoyer des signaux à d'autres entités.

**(4) Les sockets - Client/Serveur** La STS utilise un système centralisé client/serveur où toutes les entités communiquent avec un serveur central. Ce serveur central pourrait agir comme un hub pour la communication entre les différents composants du système, fournissant une architecture centralisée.

## INSTRUCTIONS

Votre mission est d'écrire un programme java complet, où les compléter les lignes de codes disponibles dans le répertoire du projet (IFT630_Projet_2) et d'implémenter un système permettant une gestion optimale de la

[1] Wikipédia.

communication inter-processus.

# Problème 1: Utilisation des mémoires partagées

## (1) Optimisation de l'efficacité du service en évitant les arrêts superflus lorsque le bus est plein.

Les entités du système de transport de la STS utilisent **des mémoires partagées** dans les IPC pour permettre à un **arrêt** de transmettre aux **bus** desservant son **trajet** des informations (*stopMessageInformation*) en temps réel sur la quantité de passagers présents à cet arrêt. Ainsi, un bus ne décide de s'arrêter à cet arrêt que s'il dispose de suffisamment de places disponibles pour accueillir une partie ou la totalité des passagers présents. Écrire le code java qui permet au stop d'envoyer les messages aux bus. Vérifier avec les bus à la réception des messages qu'ils sont les destinataires du message et font le travail attendu.

### Spécifications techniques

- Utiliser les threads pour permettre une exécution asynchrone des différentes entités du système.
- Utiliser des pools de threads pour améliorer l'efficacité de votre système en évitant la création excessive de threads et en réutilisant les threads existants.
- Utiliser les structures de données appropriées pour stocker et échanger les informations sur la quantité de passagers présents à un arrêt et dans le bus, ainsi que pour gérer les places disponibles dans les bus.
- Utilisation de mécanismes de synchronisation, tels que les verrous (locks) ou les sémaphores, pour assurer un accès sécurisé et cohérent aux données partagées entre les arrêts et les bus.

### Contraintes

- Respect des normes de sécurité et de confidentialité des données des passagers. (Encapsulation)
- Optimisation des performances pour assurer une réactivité en temps réel lors de la mise à jour et de la consultation des informations sur les passagers et les places disponibles.
- Compatibilité avec les autres composants du système de transport de la STS.

### Critères d'évaluation

- **Fiabilité** : Capacité du système à fournir des informations précises sur la disponibilité des places dans les bus en temps réel.
- **Efficacité** : Mesure de l'efficacité du système à optimiser le temps de trajet des passagers en évitant les arrêts inutiles des bus.
- **Scalabilité** : Capacité du système à gérer efficacement une augmentation du nombre de passagers et de bus sur le réseau.
- Propreté du code, gestion des erreurs et des commentaires appropriés.

# Problème 2: Utilisation des bus de messages

## (1) Gestion des horaires de bus via le modèle producteur-consommateur avec BlockingQueue.

Dans le système de transport en commun, les stations d'arrêt sont équipées de panneaux d'affichage indiquant les informations sur le bus (*BusMessageInformation*) numéro du bus, capacité du bus et l'heure d'arrivée des bus. Lorsqu'un bus quitte une station après le débarquement des passagers, il informe le prochain arrêt à son itinéraire de son horaire d'arrivée prévu. Pour gérer cet échange de messages de manière efficace, le système utilise le modèle producteur-consommateur via des *BlockingQueue*. Les bus agissent en tant que producteurs, générant des messages de type *BusMessageInformation* contenant les horaires d'arrivée et les publiant dans une *BlockingQueue*. De l'autre côté, les stations d'arrêt agissent en tant que consommateurs, lisant ces messages à partir de la *BlockingQueue* et les affichant sur les panneaux d'affichage appropriés. Attention, les arrêts n'affichent que les messages qui leur sont destinés. Afin d'assurer le bon fonctionnement du système, des mécanismes de gestion des erreurs et de synchronisation sont mis en place pour garantir que les messages sont traités de manière fiable et cohérente, contribuant ainsi à une expérience de voyage fluide pour les passagers.

### Spécifications techniques

- Utiliser la classe *BlockingQueue* pour gérer les échanges de messages entre les producteurs (bus) et les consommateurs (stations d'arrêt).
- Utiliser des pools de threads pour améliorer l'efficacité de votre système en évitant la création excessive de threads et en réutilisant les threads existants.
- Gestion des erreurs et des exceptions pour assurer la fiabilité du système.

### Contraintes

- Respect des normes de sécurité et de confidentialité des données (Encapsulation).
- Sécurité des données pour garantir que seuls les bus autorisés peuvent publier des horaires dans la BlockingQueue et que seuls les panneaux d'affichage appropriés peuvent lire et afficher ces horaires.
- Compatibilité avec les autres composants du système de transport de la STS.

### Critères d'évaluation

- **Fiabilité** : Mesure de la capacité du système à publier et à afficher les horaires de manière fiable, en évitant les erreurs.
- **Performance** : Évaluation de la réactivité du système en termes de temps de réponse lors de la publication et de l'affichage des horaires de bus.
- **Scalabilité** : Capacité du système à gérer efficacement un grand nombre de bus et d'arrêts sans compromettre les performances.
- Propreté du code, gestion des erreurs et des commentaires appropriés.

## Problème 3: Utilisation des Signaux

### (1) Gestion des horaires de bus via le modèle producteur-consommateur avec BlockingQueue.

Le système de transport en commun souhaite mettre en place un mécanisme permettant aux passagers de signaler au bus leur présence à l'arrêt et de descendre du bus une fois arrivé à destination. Pour ce faire, le système utilise une combinaison de la classe **java.util.concurrent.Future** et des méthodes *wait() et notify()*.

Lorsqu'un passager attend à un arrêt de bus, il crée un objet *Future* représentant l'arrivée du bus à cet arrêt. Pendant ce temps, le thread passager attend que ce *Future* soit terminé, indiquant ainsi que le bus est arrivé. Une fois que le passager reçoit le signal que le bus est là, il monte à bord du bus.

De l'autre côté, le thread bus, lorsqu'il arrive à un arrêt, notifie tous les passagers attendant à cet arrêt en utilisant *notifyAll()*. Les passagers, qui ont été bloqués en attendant que le bus arrive, sont maintenant libérés et peuvent monter à bord du bus.

Une fois à bord, chaque passager notifie le bus lorsqu'il souhaite descendre. Le bus, qui est en attente de ces notifications, s'arrête à l'arrêt demandé par le passager, permettant ainsi au passager de descendre en toute sécurité.

### Spécifications techniques

- Utilisation de java.util.concurrent.Future
- Les passagers créent des objets Future pour représenter l'arrivée du bus à l'arrêt.
- Les passagers attendent que le Future soit terminé pour monter à bord du bus.
- Utilisation de *wait()* et *notify()*
- Le bus notifie tous les passagers attendant à l'arrêt en utilisant *notifyAll()* dès son arrivée.
- Les passagers attendent sur un objet de verrouillage jusqu'à ce qu'ils reçoivent le signal du bus pour monter à bord.
- Les passagers notifient le bus lorsqu'ils souhaitent descendre.

### Contraintes

- Respect des normes de sécurité et de confidentialité des données (Encapsulation).
- Assurer que le mécanisme de communication entre les passagers et le bus fonctionne de manière fiable.
- Éviter les goulots d'étranglement pour assurer une exécution efficace du système, même lors de situations de charge élevée.
- Compatibilité avec les autres composants du système de transport de la STS.

### Critères d'évaluation

- **Fiabilité** : Mesure de la capacité du système à assurer la réception fiable des signaux de bus par les passagers et des notifications de descente par les passagers au bus.
- **Performance** : Mesure des temps d'attente des passagers à l'arrêt et des bus à chaque arrêt pour garantir une efficacité optimale du système.
- **Scalabilité** : Évaluation de la scalabilité du système pour supporter un nombre croissant de passagers et de bus tout en maintenant de bonnes performances.

## Problème 4: Les sockets

### (1) Optimisation de l'achalandage des passagers dans les transports en commun grâce à la communication par sockets

Le système de transport en commun souhaite mettre en place un mécanisme de communication entre la centrale et les bus pour gérer efficacement l'achalandage des passagers sur les différents trajets. Pour ce faire, la centrale agit en tant que serveur et les bus agissent en tant que clients, établissant une communication bidirectionnelle via des sockets.

Lorsque la centrale détecte qu'un trajet à plus de passagers que la capacité totale des bus sur ce trajet, elle envoie un message aux bus concernés pour les informer de la situation. Ce message contient des informations sur les trajets surchargés et recommande aux bus ayant une capacité disponible de se rediriger

vers ces trajets.

De leur côté, les arrêts (clients, stops) sont responsables d'informer la centrale du nombre de passagers présents dans les différents trajets. Ils envoient périodiquement des mises à jour à la centrale pour maintenir des données précises sur l'achalandage des passagers.

## Spécifications techniques

- Utilisation de sockets TCP/IP
- La communication entre la centrale (serveur) et les bus (clients) se fait à l'aide de sockets TCP/IP pour assurer une transmission fiable des données.
- Définition d'un protocole de communication clair entre la centrale et les bus pour l'échange d'informations sur l'achalandage des passagers et les trajets surchargés.
- Mise en place de structures de données pour stocker et mettre à jour les informations sur l'achalandage des passagers dans les différents trajets.

## Contraintes

- Respect des normes de sécurité et de confidentialité des données (Encapsulation).
- Assurer que le mécanisme de communication entre la centrale et le bus fonctionne de manière fiable.
- Éviter les goulots d'étranglement pour assurer une exécution efficace du système, même lors de situations de charge élevée.
- Compatibilité avec les autres composants du système de transport de la STS.

## Critères d'évaluation

- **Fiabilité** : Mesure de la fiabilité du système en termes de transmission réussie des messages entre la centrale et les bus, ainsi que la précision des informations sur l'achalandage des passagers.
- **Performance** : Mesure des temps d'attente des passagers à l'arrêt et des bus à chaque arrêt pour garantir une efficacité optimale du système.
- **Efficacité** : Évaluation de l'efficacité globale du système en termes de répartition équilibrée des passagers entre les différents trajets et de réduction de la surcharge dans les bus.
- **Scalabilité** : Évaluation de la scalabilité du système pour supporter un nombre croissant de passagers et de bus tout en maintenant de bonnes performances.