1. **Différence entre famine et interblocage :**

    - **Interblocage (Deadlock) :** C'est une situation où un ensemble de processus est bloqué car chaque processus attend une ressource détenue par un autre processus dans le même ensemble. Cela crée une impasse, où aucun processus ne peut progresser.
    - **Famine :** La famine (ou starvations) se produit lorsque certains processus ou threads ne peuvent pas accéder aux ressources nécessaires, même s'ils ne sont pas dans un état de blocage complet. Cela peut se produire en présence de politiques d'allocation de ressources qui ne garantissent pas un accès équitable à toutes les entités en attente.

2. **Système sans interblocage avec 3 activités et 4 ressources :**

    - Dans un système avec 3 activités et 4 ressources équivalentes, où chaque activité demande une ressource à la fois mais ne peut pas demander plus de deux ressources, on peut montrer qu'il n'y a pas d'interblocage en utilisant le "cercle d'attente". Pour qu'il y ait un interblocage, il devrait exister un cercle d'attente cyclique de processus, ce qui signifie que chaque processus dans le cercle attend une ressource détenue par le prochain processus dans le cercle.

3. **Différence entre exclusion mutuelle et transaction :**

    - **Exclusion Mutuelle :** C'est un concept dans les systèmes concurrents où l'accès à une ressource partagée est accordé à un seul processus à la fois. Cela garantit qu'aucun autre processus n'entre en conflit pendant l'accès à la ressource partagée. Les sections de code en exclusion mutuelle sont souvent implémentées à l'aide de mécanismes tels que les verrous pour éviter les conflits d'accès.

    - **Transaction :** Une transaction est une unité logique d'exécution dans un système de gestion de base de données (SGBD). Elle représente une série d'opérations qui doivent être exécutées de manière atomique, c'est-à-dire que toutes les opérations de la transaction doivent être exécutées avec succès, sinon aucune d'entre elles ne doit être effectuée. Les transactions garantissent la cohérence des données et la préservation de l'intégrité de la base de données, même en cas d'échec.

En résumé, l'exclusion mutuelle se concentre sur l'accès sécurisé aux ressources partagées dans un contexte de programmation concurrente, tandis que les transactions sont utilisées dans les bases de données pour garantir l'intégrité des données dans le contexte de l'accès et de la modification des données.

. Problème avec le code :

- Le problème avec le code est qu'il présente un risque de deadlock. Si les activités concurrentes obtiennent les verrous dans un ordre différent, il peut se produire un deadlock où chaque activité attend un verrou détenu par une autre activité, créant une impasse où aucune activité ne peut progresser.

2. Solutions pour corriger le problème :

- **Solution 1 - Ordre d'acquisition des verrous :** Imposer un ordre d'acquisition des verrous pour éviter le deadlock. Par exemple, chaque activité doit acquérir les verrous dans le même ordre (par exemple, lockx, locky, lockz) pour garantir l'absence de deadlock.

- **Solution 2 - Utilisation de l'instruction tryLock :** Utiliser des instructions de verrouillage qui permettent de tenter l'acquisition d'un verrou sans bloquer le processus en cas d'échec. Cela peut être réalisé avec des méthodes telles que tryLock() pour éviter les deadlock en vérifiant si le verrou peut être obtenu avant de l'acquérir.

3. Intérêt des pools de threads :

- Les pools de threads offrent plusieurs avantages par rapport à la création à la demande des activités :
    - **Réutilisation des threads :** Les pools de threads permettent de réutiliser les threads existants au lieu de créer et détruire des threads à chaque fois, ce qui est plus efficace en termes de performance.
    - **Gestion des ressources :** Les pools de threads gèrent automatiquement le nombre de threads actifs, évitant ainsi une utilisation excessive des ressources système.
    - **Contrôle sur la concurrence :** Les pools de threads offrent un contrôle plus fin sur la concurrence, en permettant de limiter le nombre de threads actifs simultanément, ce qui peut être important pour éviter la surcharge du système.

4. Parallélisation de la recherche dans un tableau d'entiers non trié :

- Pour paralléliser la recherche dans un tableau d'entiers non trié, le schéma fork/join serait généralement préférable. Il permet de diviser le problème en sous-problèmes plus petits, les traiter de manière récursive de manière parallèle, puis agréger les résultats. Cela s'aligne bien avec la nature de la recherche dans un tableau non trié, où les sous-sections du tableau peuvent être explorées de manière indépendante. Un pool de threads pourrait également être utilisé dans le contexte fork/join pour gérer la parallélisation.

wait tests if any other process using the ressource and if not it will decrement S, if it's using it the process wil be stuck in a while loop.

signal is to say that i completed using the semaphore you re free to use it

---

Famine (Starvation) :

Définition : La famine se produit lorsqu'une entité (un processus, un thread, etc.) est empêchée de progresser ou d'obtenir une ressource nécessaire malgré ses tentatives répétées.

Cause : La principale cause de la famine est souvent due à une mauvaise gestion des priorités ou à un accès inéquitable aux ressources. Certains entités peuvent être continuellement contournées par d'autres entités avec des priorités plus élevées.

Effet : Une entité en famine peut être inactive indéfiniment, ce qui peut entraîner un comportement indésirable dans le système.

Interblocage (Deadlock) :

Définition : L'interblocage se produit lorsqu'un ensemble de processus ou de threads est bloqué, chacun attendant qu'une ressource détenue par un autre membre de l'ensemble soit libérée.

Cause : L'interblocage se produit généralement lorsque des entités concurrentes demandent des ressources de manière cyclique et les maintiennent pendant que chacune d'elles attend une ressource détenue par une autre.

Effet : L'interblocage peut conduire à une impasse complète où aucun des processus ou threads ne peut progresser, ce qui peut nécessiter une intervention externe pour résoudre la situation.

Exclusion Mutuelle :

Objectif : L'exclusion mutuelle vise à éviter que plusieurs threads ou processus n'accèdent simultanément à une section critique du code, ce qui pourrait entraîner des conflits et des résultats inattendus.

Mécanisme : Un mécanisme tel que les verrous (locks) est souvent utilisé pour mettre en œuvre l'exclusion mutuelle. Lorsqu'un thread entre dans la section critique, il acquiert le verrou, empêchant ainsi d'autres threads d'entrer dans la même section critique jusqu'à ce que le verrou soit libéré.

Portée : L'exclusion mutuelle est souvent utilisée pour gérer l'accès à des ressources partagées telles que des variables, des structures de données ou des fichiers.

Pour montrer que le système décrit est nécessairement exempt d'interblocage, nous pouvons utiliser la méthode du graphe d'allocation de ressources.

Voici comment nous pouvons démontrer l'absence d'interblocage dans ce système :

Identifier les ressources et les activités :

Le système a 4 ressources équivalentes.

Il y a 3 activités qui peuvent demander une ressource à la fois, mais ne peuvent pas demander plus de deux ressources.

Construire le graphe d'allocation de ressources :

Chaque nœud dans le graphe représente une activité ou un processus.

Chaque flèche représente une demande de ressource.

Chaque ligne représente une ressource.

Vérifier l'absence de cycle dans le graphe :

Un interblocage se produit lorsqu'il y a un cycle dans le graphe d'allocation de ressources.

Dans ce cas, chaque activité demande une ressource à la fois, et il y a suffisamment de ressources pour satisfaire chaque demande individuelle. Par conséquent, il ne peut pas y avoir de cycle dans le graphe d'allocation de ressources

---

Le modèle fork/join est un paradigme de programmation parallèle qui est particulièrement adapté aux problèmes récursifs où une tâche peut être décomposée en sous-tâches indépendantes. Ce modèle est souvent implémenté au moyen d'un framework qui automatise la gestion de la concurrence et la distribution du travail entre plusieurs threads. L'une des implémentations les plus connues de ce modèle est le framework Fork/Join introduit dans Java à partir de la version 7.

Les concepts clés du modèle Fork/Join sont les suivants :

Fork : La tâche principale se divise en sous-tâches plus petites (fork), créant ainsi une hiérarchie de tâches. Chaque sous-tâche est attribuée à un thread disponible.

Join : Les résultats des sous-tâches sont combinés (join) pour obtenir le résultat global de la tâche. Les threads attendent que les sous-tâches soient terminées avant de combiner les résultats.

Work Stealing : Les threads peuvent "voler" du travail des autres threads lorsqu'ils n'ont plus de tâches à exécuter. Cela permet une utilisation plus efficace des threads disponibles.

L'objectif du modèle Fork/Join est de tirer parti de la parallélisme disponible dans le problème à résoudre en le décomposant en sous-problèmes qui peuvent être résolus indépendamment. Cela est particulièrement adapté aux tâches récursives où une tâche peut être divisée en plusieurs sous-tâches.