Projet de Programmation Système :
Implémentation d'un Allocateur Mémoire

Objectif du Projet

L'objectif de ce projet est d'implémenter un allocateur mémoire personnalisé en utilisant un langage bas niveau (C, C++ ou Rust). Vous allez développer une version basique d'un allocateur basé sur `mmap`, puis explorer des optimisations pour améliorer ses performances. Le but est d'obtenir une solution fonctionnelle qui se rapproche des performances de l'allocateur système (`malloc`/`free`) tout en restant accessible.

Compétences visées

- Maîtrise des appels système (`mmap`, `munmap`)
- Compréhension de la gestion mémoire bas niveau
- Implémentation d'algorithmes d'allocation mémoire (allocation/désallocation)
- Optimisation des performances

Énoncé

Vous devez implémenter un allocateur mémoire personnalisé avec les fonctions suivantes :

- `void* my_malloc(size_t size)` : alloue un bloc mémoire de taille `size` octets.
- `void my_free(void* ptr)` : libère le bloc mémoire pointé par `ptr`.

Contraintes Initiales :

1. Utiliser `mmap` pour obtenir de la mémoire (chaque appel à `my_malloc` doit faire appel à `mmap`).
2. Implémenter une gestion basique des blocs mémoire sans caching.
3. La solution doit fonctionner sans optimisations avancées mais viser à approcher les performances de l'allocateur système.

Étapes de réalisation

1. **Implémentation Basique**
    - Implémenter `my_malloc` en utilisant `mmap`.
    - Implémenter `my_free` en utilisant `munmap`.
    - Valider le fonctionnement de votre allocateur avec des tests unitaires.
2. **Analyse des Performances**
    - Comparer les performances avec `malloc`/`free` en mesurant le nombre d'allocations/désallocations par seconde.
    - Identifier les goulots d'étranglement.
3. **Optimisations**

- Vous pouvez améliorer votre implémentation de base en utilisant les optimisations suggérées ci-dessous.

Suggestions d'Améliorations

Voici une liste d'améliorations possibles avec leur niveau de complexité et le gain estimé en termes de performance. Chaque optimisation est notée sur une échelle de 20 points.

Amélioration	Complexité	Gain estimé	Description
1. **Segmentation en classes de tailles**	Moyenne	+3 points	Utiliser des listes de blocs pour différentes tailles (ex: puissances de 2).
2. **Recyclage des blocs libérés**	Moyenne	+2 points	Implémenter un cache de blocs précédemment alloués pour éviter les appels à `mmap`.
3. **Coalescence des blocs libres**	Élevée	+4 points	Fusionner les blocs adjacents libérés pour réduire la fragmentation.
4. **Allocation best fit**	Faible	+1 point	Rechercher le bloc libre le plus proche en taille pour réduire le gaspillage.
5. **Gestion multi-thread avec locks**	Moyenne	+2 points	Ajout de support multi-thread avec des `mutex` pour protéger l'allocateur.
6. **Cache par thread**	Élevée	+4 points	Fournir un cache mémoire par thread pour limiter les locks globaux.
7. **Alignement des blocs mémoire**	Faible	+1 point	Assurer l'alignement des blocs pour des performances optimales sur certaines architectures.
8. **Détection des fuites mémoire**	Moyenne	+2 points	Ajouter un suivi des allocations pour détecter les fuites mémoire (bonus).
9. **Allocation par arène**	Élevée	+3 points	Créer plusieurs arènes pour limiter la contention en multi-thread.
10. **Optimisation des métadonnées**	Faible	+1 point	Réduire la taille des en-têtes des blocs pour économiser de la mémoire.

Consignes de Remise

- **Deadline** : 13 décembre.
- **Remise** par email à `jbbesnard@paratools.com` avec comme objet "**AISE25 ALLOC [Nom Binôme]**".
- Le fichier `README.md` est **obligatoire** et doit contenir :
  - Instructions de compilation et d'exécution.
  - Explication des choix d'implémentation.
  - Liste des points d'optimisations (mettez en valeur vos travaux!)
  - Résultats des benchmarks (si réalisés).
- Un **rapport** (PDF) est **optionnel** mais sera pris en compte pour des points bonus.

Barème (sur 20 points)

- Implémentation de base : 7 points
- Tests et validation : 3 points
- Optimisations : 7 points
- Qualité du code & documentation (`README.md`) : 3 points
- Bonus (si rapport ou optimisations avancées) : jusqu'à +2 points