Projet : Gestion des tâches à l'aide de listes chaînées

Contexte

Dans certaines applications ou systèmes informatiques, il est essentiel de suivre et gérer l'allocation des ressources (comme la mémoire et le temps de calcul) occupées par des tâches à effectuer. Lorsque ces tâches sont des déploiements de scripts, des exécutions de tests ou des jobs de calcul, il faut respecter les exigences de chaque tâche en termes de ressources. Une gestion efficace de ces ressources permet non seulement d'assurer la bonne exécution des tâches, mais aussi d'optimiser l'utilisation de l'infrastructure disponible.

Objectif du projet

Ce projet vise à implémenter un gestionnaire de tâches de calcul utilisant des listes chaînées, permettant ainsi de manipuler et organiser dynamiquement des éléments tout en maîtrisant la gestion de la mémoire, un aspect crucial en développement logiciel. L'application permettra aussi d'allouer des tâches à des serveurs disponibles.

L'objectif est de créer une application en ligne de commande qui gère une liste de tâches. Chaque tâche devra comporter :

- Un identifiant du processus
- Une quantité de RAM requise
- Un temps de calcul estimé
- Un état ("à faire", "en cours", "terminée")

Les fonctionnalités à implémenter incluent l'ajout, la suppression, le tri et la mise à jour des tâches en manipulant une liste chaînée. Une analyse des performances de votre algorithme de tri devra être réalisée**.**

Exercice 1. **Créer une tâche et l'ajouter à la liste**

Proposez une fonction *addTask* permettant de créer une nouvelle tâche avec un titre, une description, une priorité et un état. La tâche sera ajoutée à la fin de la liste chaînée.

Objectif : Manipuler les éléments d'une liste chaînée et ajouter de nouveaux nœuds.

Exercice 2. **Supprimer une tâche par son titre**

Proposez une fonction *deleteFromId* permettant de supprimer une tâche en utilisant son titre comme identifiant unique. Si la tâche existe dans la liste, elle doit être supprimée et la liste doit être mise à jour correctement.

Objectif : Travailler sur la suppression d'un élément dans une liste chaînée et gérer les pointeurs des éléments voisins.

Exercice 3. Modifier l'état d'une tâche

Proposez une fonction *updateState* permettant de modifier l'état d'une tâche en fonction de son titre. L'état peut être modifié en une des trois valeurs suivantes : "à faire", "en cours", "terminée".

Objectif : Manipuler les données dans la liste chaînée et effectuer une mise à jour sur un nœud spécifique.

Exercice 4. **Trier les tâches par priorité (en place)**

Proposez une fonction *sortTasks* permettant de trier les tâches par priorité (du plus important au moins important). Utilisez un algorithme de tri adapté aux listes chaînées, en évitant de créer de nouvelles listes ou de nouveaux éléments (tri en place). Chaque groupe implémente un algorithme de tri différent parmi les suivants :

- Tri par insertion
- Tri fusion
- Tri rapide (Quicksort, adapté aux listes chaînées)
- Tri par sélection
- Tri cocktail

Objectif : Apprendre à trier une liste chaînée en manipulant les pointeurs des nœuds, sans allouer de mémoire supplémentaire.

Exercice 5. **Insérer une tâche à la place correspondant à sa priorité**
Proposez une fonction orderedInsert qui prend en entrée une liste chaînée triée par priorité et une nouvelle tâche et qui insère la nouvelle tâche en l'interclassant entre tâches de la liste de façon à maintenir la liste triée.

Objectif : Maintenir organisée une collection d'éléments.

Exercice 6. Générer des éléments aléatoires pour la liste

Implémentez une fonction *generateRandomTasks* générant un nombre donné de tâches avec des titres aléatoires, des priorités aléatoires (exemple : de 1 à 10), et des états aléatoires ("à faire", "en cours", "terminée"). Cette fonction sera utilisée pour tester vos algorithmes de tri et pour comparer leur efficacité sur des jeux de données variés.

Objectif : Créer des données de test pour permettre la comparaison des algorithmes de tri et leur performance.

Exercice 7. **Analyse des performances des algorithmes de tri**

Créez un tableau indiquant les temps d'exécution d'une procédure permettant d'obtenir une liste triée de n tâches générées aléatoirement, pour n = 10, 100, 1000, 10000. Tracez la courbe correspondante et analysez si elle suit la complexité théorique que vous aviez imaginée pour votre fonction.

Objectif : Expérimenter et analyser les performances empiriques des algorithmes de tri en les comparant à leur complexité théorique.

Exercice 8. **Un serveur vient de se libérer**

Un serveur vient de se libérer et dispose d'une certaine quantité de RAM. Implémentez une fonction *allocate* permettant de sélectionner et d'affecter à ce serveur les tâches de la liste en maximisant son remplissage tout en respectant sa capacité mémoire.

Objectif : Concevoir un algorithme d'allocation efficace optimisant l'utilisation des ressources disponibles.

Conclusion

Ce projet permet de développer des compétences en manipulation des listes chaînées, en gestion de la mémoire dynamique et en optimisation algorithmique. En expérimentant des stratégies de tri et en analysant leurs performances, de même qu'en résolvant un problème d'optimisation, ce projet offre une meilleure compréhension des structures de données et des algorithmes fondamentaux utilisés en programmation.