NF16 – TP03

Compte-rendu

Listes chaînées

Auteurs : Akim Sadaoui et Antoine Hars

- **Introduction :**

Le but de ce tp est de développer une application simplifiée pour la gestion d'une série de tâches à accomplir par un processeur appelé « Colombo ».
À travers ce tp, nous avons pu apprendre à manipuler des listes simplement chaînées.
Le principe de ce tp consiste à modéliser une liste de tâches à exécuter, l'exécution étant représentée par l'instruction sleep(1), et à manipuler la liste de tâches.

Une tâche correspond à une structure comprenant un identifiant, une durée, une priorité pour l'exécution ainsi qu'un pointeur sur la tâche qui la suit dans la liste.

Pour accéder à la liste de tâches, nous avons décidé d'utiliser une sentinelle qui pointera toujours sur la première tâche de la liste. La dernière tâche de la liste pointera, quant à elle, sur la sentinelle.

- **Les fonctions de base :**

Nous avons regroupé dans cette partie, toutes les fonctions permettant la manipulation des tâches et des listes de tâches.

La fonction de création de tâche de la forme :

- *task * cree_tache (char carac[MAX_NOM+1], int duree)*

Cette fonction nous permet de créer dynamiquement une tache avec pour identifiant *carac* et pour durée *duree*. La priorité de la tâche est instanciée à *duree /* 10 et le pointeur pointant vers la tâche suivante (servant pour l'instanciation d'une liste de tâches) est initialisé à NULL.

La fonction de création de liste de la forme :

- *task * cree_liste (task *tache)*

Cette fonction de création de liste nous donne la possibilité de créer une chaîne simple de tâches. Dans cette fonction, nous créons la sentinelle qui va pointer vers tâche ou vers null si nous voulons une liste vide.

La fonction d'affichage de la liste :

- *void affiche_liste (task *list_task)*

La fonction d'affichage de la liste effectue en fait un parcours de la liste de tâches et affiche l'identifiant et la durée de chaque élément de la liste (excepté la sentinelle).
Le parcours de la liste est obtenu par le pointeur de chaque tâche pointant sur son suivant jusqu'à atteindre la sentinelle.

Vu que la liste de n éléments est entièrement parcourue afin d'afficher tous les éléments de la liste, la complexité de cette fonction est donc en O(n).

La fonction d'ajout de tâche à la liste de la forme :

- *int ajoute_tache (task *list_task, task *ptache)*

Cette fonction d'ajout de tâche effectue un parcours complet de la liste de tâches pour insérer *ptache* à la fin de cette liste. Le dernier élément de la liste ne pointera plus sur la sentinelle mais sur *ptache* et le pointeur *psuivant* de *ptache* pointera donc vers la sentinelle. La fonction renvoie 0 en cas de succès de l'ajout et 1 en cas d'échec.

Étant donné que la liste de n éléments est entièrement parcourue pour ajouter la tâche à la fin de la liste, la complexité peut être représentée en 0(n).

La fonction de recherche de la position d'une tâche dans la liste :

- *int search_ID (task *list_task, char carac[MAX_NOM+1])*

Cette fonction va rechercher pour l'identifiant donné *carac*, la position de la tâche correspondante dans la liste. Si la première occurrence de l'identifiant de la tâche est trouvée, la fonction renvoie la position de la tâche dans la liste sinon elle renvoie -1.

Vu que pour trouver la tâche dans la liste de n éléments, il est possible que nous parcourions la totalité de la liste, la complexité de cette fonction est en O(n) dans le pire des cas. Dans le meilleur des cas, elle est en Ω(1).

La fonction d'annulation d'une tâche de la liste :

- *task * annule_tache (task *list_task, char carac[MAX_NOM+1])*

La fonction d'annulation d'une tâche va faire appel à la fonction *search_ID* pour trouver la position de la tâche dans la liste si elle est présente dedans.
Si la tâche est présente, l'annulation de la tâche va consister à faire pointer le pointeur de la tâche précédente vers la tâche suivante de celle à supprimer et enfin à libérer l'espace mémoire alloué à cette tâche (*free(tache)*).

Pour accéder à la tâche, il y a un parcours de la liste jusqu'à la position de la tâche à supprimer donc la complexité de la fonction est en O(n) dans le pire des cas pour une liste de n éléments (si la tâche est en dernière position de la liste). Dans le meilleur des cas, la tâche est en première position, donc a une complexité en Ω(1).

La fonction qui supprime toute la liste de tâches :

- *task * libere_liste (task *list_task)*

Cette fonction de libération de la liste de tâche permet de libérer l'espace mémoire alloué à chaque élément de la liste. La liste retourne à la fin sa sentinelle seule.
La fonction parcourt tous les éléments de la liste et les supprime à l'aide de la fonction *free(tâche)*.

Vu que la fonction parcourt toute la liste pour en supprimer tous les éléments, exceptée la sentinelle, la complexité de *libere_liste* est en O(n) dans tous les cas de figure.

- **Le fonctionnement du processeur :**

Le fonctionnement du processeur en mode FIFO :

- *task * execute_tache_FIFO (task *list_task)*

Cette fonction d'exécution en mode FIFO permet d'exécuter la première tâche de la liste (principe de la file d'exécution). Cette tâche sera ensuite retiré de la liste de tâches et la fonction renvoie la liste de tâches sans la tâche exécutée. Le pointeur *psuivant* de la sentinelle aura pour valeur l'adresse de la seconde tâche de la liste.
L'exécution de la tâche est caractérisée par l'utilisation de la fonction *sleep(1)* qui permet au programme de faire une pause d'une seconde pour symboliser une exécution de tâche.

La fonction d'exécution en mode FIFO ne fait appel qu'à des instructions simples que nous travaillons sur la première tâche de la liste (aucun besoin de la parcourir) donc la complexité de cette fonction est en O(1).

Le fonctionnement du processeur en mode LIFO :

- task * execute_tache_LIFO (task *list_task)

La fonction d'exécution en mode LIFO permet d'exécuter la dernière tâche de la liste (principe de la pile d'exécution). Nous devons donc faire un parcours de la liste dans un premier temps pour avoir la dernière tâche ajoutée. L'exécution de la dernière tâche entraîne sa libération de la liste.

Pour obtenir la dernière tâche de la liste qui sera exécutée, nous devons parcourir la liste de n éléments obligatoirement donc dans le meilleur des cas, la liste ne contient qu'une tâche et la complexité est en Ω(1) sinon dans le pire des cas, la complexité est en O(n)

Le fonctionnement du processeur en mode FIFO, avec priorité selon la durée des tâches :

- *task * insere_tache (task *list_task, task *ptache)*

Cette fonction insère dans une liste une tâche en fonction de la durée. Elle parcourt la liste jusqu'à ce que la valeur de la durée de la tâche soit inférieur à la durée d'une tâche de la liste pour ensuite insérer la tâche en paramètre à l'endroit obtenu grâce au parcours de la liste.

Dans le meilleur des cas, la liste est vide ou la durée de la tâche est inférieure à la première tâche de la liste et la complexité est en Ω(1) sinon dans le pire des cas, la tâche est insérée à la fin de la liste et donc la complexité est en O(n).

- *task * load_data (char * nom_fichier, int nb_taches)*

Cette fonction permet de charger un ensemble de nb_taches tâches dans une liste.

Dans le meilleur des cas, on ne charge qu'une seule tâche dans la liste et la complexité est en Ω(1) et dans le pire des cas, nous insérons toutes les tâches du fichier, donc la complexité est en O(n).

- *task * fusion_listes (task *list_task1, task * list_task2)*

Cette fonction nous permet de fusionner 2 listes de tâches triées selon leur durée. Nous parcourons donc les 2 listes pour insérer la tâche avec la plus petite durée dans une nouvelle liste.

La complexité dans le meilleur des cas (celui où les 2 listes sont vides) est en Ω(1) et la complexité dans le pire des cas, celui où la première liste a n1 éléments et la seconde n2 éléments est en O(n1 + n2) car on parcours n1 + n2 éléments. Cela revient à avoir une complexité en O(n).

Le fonctionnement du processeur en mode FIFO, avec changement dynamique de priorité :

- *task * insere_tache_priorite (task *list_task, task *ptache)*

Cette fonction nous permet d'insérer une tache dans la liste en fonction de sa priorité par rapport à celle de chacune des tâches de la liste. On parcourt la liste jusqu'à trouver une tâche de la liste ayant sa priorité supérieur à celle de la tâche à insérer.

Dans le meilleur des cas, nous insérons la tâche en début de liste (sans avoir à la parcourir) donc la complexité est en Ω(1) sinon dans le pire des cas, elle est en O(n) car nous devons parcourir toute la liste pour insérer la tâche à la fin.

- *int MAJ_priorite (task *list_task)*

Cette fonction parcourt la liste pour décrémenter chaque tâche après une exécution en mode FIFO de la liste (exécution de la première tâche de la liste).

Vu que nous parcourons toute la liste de tâches pour décrémenter la valeur de la priopriété, la complexité de la fonction est en O(n).

- **Conclusion :**

La réalisation de ce tp nous a permit d'apprendre à mieux manier une structure de liste simplement chaînée avec les fonctions d'ajout, de suppression, de recherche d'éléments.
Ce tp nous a permit aussi à simuler l'exécution en mode FIFO et LIFO vu en cours, et donc à nous familiariser avec ces notions ainsi qu'avec l'utilisation d'une sentinelle pour récupérer le premier élément d'une liste.