TP7/8: Stratégies d'insertion mémoire

Le but de cet exercice est d'évaluer expérimentalement l'efficacité des 3 stratégies d'insertion de blocs de données en mémoire vues en cours (First-fit, Best-fit et Worst-fit). L'évaluation se fera selon deux critères : l'efficacité mémoire, et le temps d'exécution.

Pour cela, nous représentons la liste de blocs libres sous forme de liste simplement chaînée. Chaque maillon de la liste représente un bloc mémoire libre et est implémenté en C par la structure suivante :

```
struct bloc_libre_s {
   int taille_bloc;
   struct bloc_libre_s *suivant;
};
typedef struct bloc_libre_s bloc_libre;
```

Partie I : Implémentation des stratégies d'insertion

Question 1

Implémenter la fonction `bloc_libre *generer_liste_aleatoire(int nb_blocs)` qui crée une liste de `nb_blocs` blocs libres. La taille de chaque bloc est un entier généré aléatoirement (avec la fonction `rand()`) entre 16 et 1024.

Question 2

Implémenter la fonction `void afficher_blocs_libres(bloc_libre * liste)` qui affiche dans la console la liste des blocs libres passée en paramètre.

Testez les fonctions `generer_liste_liste_aleatoire` et `afficher_blocs_libres`.

Question 3

Implémenter la fonction `int calculer_espace_libre(bloc_libre * liste)` qui parcourt la liste pour calculer l'espace mémoire libre total (autrement dit la somme des tailles de tous les blocs libres).

Question 4

Implémenter la fonction `int inserer_first_fit(int taille_bloc, bloc_libre *liste)` qui insère une donnée de taille `taille_bloc` dans la liste de blocs libres en utilisant la stratégie **First fit**. La fonction renvoie -1 si l'insertion est un échec, 0 sinon.

Question 5

Implémenter la fonction `int inserer_best_fit(int taille_bloc, bloc_libre *liste)` qui insère une donnée de taille `taille_bloc` dans la liste de blocs libres en utilisant la stratégie **Best fit**. La fonction renvoie -1 si l'insertion est un échec, 0 sinon.

Question 6

Implémenter la fonction `int inserer_worst_fit(int taille_bloc, bloc_libre *liste)` qui insère une donnée de taille `taille_bloc` dans la liste de blocs libres en utilisant la stratégie **Worst fit**. La fonction renvoie -1 si l'insertion est un échec, 0 sinon.

Partie II : Évaluation des stratégies d'insertion

Nous allons maintenant tester l'efficacité des stratégies d'insertion. On propose d'abord de mesurer le taux d'occupation mémoire. Pour cela, on procède comme suit :

- on crée une liste de 20000 blocs libres en utilisant la fonction `generer_liste_aleatoire`
- on calcule l'espace libre total avec la fonction `calculer_espace_libre`.
- on insère des données de taille aléatoire (comprise entre 16 et 1024) dans la liste de blocs libres jusqu'à arriver à saturation de la liste (la fonction d'insertion doit retourner -1).
- On calcule (et on affiche) le taux d'occupation de la mémoire ($taux\ d'occupation = \frac{quantite\ de\ donnees\ inserees}{espace\ libre\ initial}$)

Question 7

En utilisant le protocole expérimental ci-dessus, calculer le taux d'occupation mémoire **pour les trois stratégies d'insertion**. Que constatez-vous ?

On souhaite maintenant tester le temps d'exécution de chaque stratégie. Pour cela, on propose d'utiliser la librairie de `chronometre` disponible à cette adresse : *http ://pagesperso-systeme.lip6.fr/Sergey.Legtchenko/*

Le protocole expérimental est simple :

- on crée une liste de 20000 blocs libres en utilisant la fonction `generer_liste_aleatoire`
- on insère 100 données de taille aléatoire (comprise entre 16 et 1024), tout en chronometrant le temps total mis par les stratégies pour inserer les 100 blocs.

Question 8

En utilisant le protocole expérimental ci-dessus, évaluer le temps d'exécution des 3 stratégies. Que constatez-vous ?