**TPn°2 INFORMATIQUE EMBARQUE**

RASOLOARISON Tina
BELMILOUD Samira

Estimation du temps nécessaire : 8 heures

Temps nécessaire : entre 11 heures

**INTRODUCTION**

Le problème des listes chaînées est que leur taille croît linéairement avec le nombre d'objets qui se trouvent dans la liste. Le temps de parcours (de traitement) d'une liste devient alors proportionnel au nombre d'objets de cette liste. Un moyen de minimiser ce problème est de faire appel à des hash listes (liste de hachage). C'est un outil général dans l'univers du noyau Linux.

Dans ce projet nous allons fournir:

- Une fonction d'initialisation de la liste de hachage,
- une fonction d'insertion,
- une fonction de retrait,
- et une fonction permettant de retrouver un objet à partir de son identifiant.
-

Avec ces fonctions nous allons manipuler deux types de structures :

Structure A : une structure d'objets avec un identifiant qui soit un entier strictement positif.

Structure B : une structure d'objets avec un identifiant qui soit un entier toujours multiple de 4096.

**Table de Hachage:**

Les tables de hachage constituent quelque part un compromis entre les listes chainées et les tableaux. Elles sont flexibles comme les listes chainées et permettent un accès rapide comme les tableaux.

Pour manipuler les listes chainées dans notre projet. Nous avons utilisé les macros suivantes :

1.# define offsetof (TYPE , MEMBER ) ({ ( size_t ) &(( TYPE *)0)->MEMBER })

Calcule la différence entre le pointeur sur l'élément et le pointeur sur l'objet.

2. # define container_of (ptr , type , member ) ({ const typeof ( (( type *)0)->member ) * __mptr = ( ptr); ( type *)( ( char *) __mptr - offsetof (type , member ) );})

Sert à calculer l'adresse d'un objet dont un des champs est un élément dont le type est struct list_head, à partir d'un pointeur sur cet élément, du nom de ce champ dans l'objet, et du type

de l’objet.

3. # define list_for_each_entry (cur , head , member ) for ( cur = container_of (( head )->next ,\\

typeof (*cur), member );

&cur ->member != ( head );

cur = container_of (cur ->member .next , typeof (*cur), member ))

Joue le rôle d’itérateur, en simplifiant l’écriture de la boucle for pour parcourir les éléments d’une liste de type list_head.

Fonction d'insertion

```
void list_add (struct list_head *node, struct list_head *head) {
head->next->prev = node;
node->next = head->next;
node->prev = head;
head->next = node;
}
```

On remarque dans cette fonction que l’insertion se fait en tête de la liste. Node est le nœud qu’on va insérer et head et la tête de liste.

Fonction de retrait

```
void list_del (struct list_head *node, struct list_head *head) {
if(node->next) {
head->next = node->next;
node->next->prev = head;
}
}
```