Cours "Informatique Embarquée"

François Armand

Exercice N°2, 17 Octobre 2014
À rendre avant le 23 Octobre 23H59
(Limite impérative) sur Didel

Hash Lists

1. Consignes

Les consignes habituelles s'appliquent :

- Estimation du temps, temps réel passé
- Relecture par un autre groupe
- Structure des sources, Makefile...

1. Hash Lists

1.1. Introduction, motivation

Les systèmes d'exploitation utilisent des outils génériques pour remplir leurs tâches. Un des mécanismes les plus courants est de gérer des structures de données (descripteurs de processus, de fichiers, de régions mémoires....). Cela consiste très souvent en opérations d'allocation, de libération et en recherche d'une structure donnée à partir d'un identificateur.

On a aussi parfois à placer une structure dans un ensemble existant. Par exemple, lorsqu'un processus devient éligible, il faut non seulement le "marquer" comme tel, mais aussi l'insérer dans l'ensemble des processus que l'ordonnanceur devra examiner pour trouver celui qui se verra attribuer le processeur.

Ces listes sont aussi utilisées dans le monde applicatif.

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 (quasiment une bibliothèque) dans l'univers du noyau Linux.

Dans le monde Java, on utilise aussi des hashmap, qui permettent de stocker des éléments en les associant à une clé ou un identifiant, et de les retrouver.

1.2. Listes chainées

Vous écrirez (en langage C) une bibliothèque fournissant des services de listes de gestion de listes circulaires doublement chaînées.

```
void INIT_LIST_HEAD (struct list_head *head);
void list_add (struct list_head *node, struct list_head *head);
void list_del (struct list_head *node);
list_for_each_entry(cur, head, member)
```

L'itérateur sera générique et devra permettre de parcourir des listes d'objets incluant des éléments de type `struct list_head` **quelle que soit la position de l'élément `struct_list_head` dans l'objet. Un objet doit aussi pouvoir appartenir à plusieurs listes simultanément.**

```
struct my_object {
     ....
      struct list_head my_object_listA;
     ....
     struct list_head my_object_listB;
     ....
}
```

La « complexité » réside essentiellement dans l'itérateur. Pour vous simplifier le travail, vous vous aiderez de la macro suivante :

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

Vous chercherez donc à comprendre ce que fait cette macro ainsi que les « fonctions » prédéfinies `typeof` et `offsetof`.

1.3. Table de « hash »

Sur la base de ces listes chaînées, vous créerez (en langage C) un service de table de hash permettant de manipuler deux types de structures de données:

- Structure A : vous définirez une structure d'objets avec un identifiant qui soit un entier strictement positif. Exemple : une liste d'i-nodes (descripteurs de fichiers) chaque i-node étant identifié par son numéro.
- Des chaînes de caractères.

Il n'est pas nécessaire de définir un objet contenant à la fois une structure A et une chaîne de caractères. Vous fournirez :

- 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

Il serait extrêmement raisonnable que dans un programme multi-threads, différentes threads puissent travailler de manière concurrente sur des tables de hachage différentes.

Et évidemment au-delà de ces deux « bibliothèques » (listes chaînées et tables de hash), un petit programme main exerçant ces fonctions.

La rédaction de cette deuxième partie est volontairement peu précise. Contrairement à la première partie, il n'y a pas d'API qui soit imposée. Libre à vous de proposer ce qui vous semble le mieux.