

Tables de hachage pour le programmeur C

Stéphane Bortzmeyer

<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 23 Janvier 2009. Dernière mise à jour le 9 Février 2009

<http://www.bortzmeyer.org/c-hash-tables.html>

Les tables de hachages, appelées, selon les langages de programmations, "*hashes*" (Perl), dictionnaires (Python), `HashTable` (Java), sont une des structures de données les plus courantes en programmation, en partie à cause de leur simplicité. Sont-elles réservées aux langages de haut niveau ? Et en C ?

Ce langage est parfois nécessaire par exemple pour des raisons de performance. Ayant eu à traiter de grandes quantités de données, j'ai tenté un essai en C. Mais le problème se prêtait bien aux tables de hachage et je n'en avais jamais fait en C. Allait-il falloir que je les code moi-même ? Non, il existe plusieurs bibliothèques de qualité déjà existantes, et en logiciel libre.

Voici une liste, non exhaustive (et je n'ai pas eu le temps de tout tester) :

- La GLib fournit une énorme quantité de structure de données, dont les tables de hachage. Très répandue, elle est bien documentée (<http://library.gnome.org/devel/glib/stable/glib-Hash-Tables.html>) et, je trouve, très facile à utiliser. C'est de loin celle qui m'a permis de terminer le plus vite.
- HamsterDB (<http://www.hamsterdb.com/>) est surtout connu pour sa capacité à sauvegarder les tables sur disque. Mais, pour des tables temporaires, il peut aussi fonctionner en mémoire, avec l'option `HAM_IN_MEMORY_DB`. Je le trouve moins bien documenté que la GLib mais la photo sur la page d'accueil du site Web est trop mignonne. Par contre, l'absence d'un mécanisme de tri pratique m'a fait perdre du temps.
- Judy est utilisée dans un certain nombre de programmes et semblait prometteuse. Mais la documentation est horrible et la bibliothèque est donc d'un abord difficile (sans compter les noms incompréhensibles de ses macros).
- HashIt (<http://freshmeat.net/projects/hashit/>) semble intéressant mais j'ai manqué de temps pour le tester.
- Comme HamsterDB, Berkeley DB est surtout conçu pour garder les tables sur disque mais il a apparemment une option pour rester en mémoire, décrite ainsi : « In-memory databases never intended to be preserved on disk may be created by setting the file parameter to NULL. »
- Enfin, en s'éloignant des tables de hachage, il faut aussi signaler que SQLite a aussi un mode « en mémoire » (`base":memory:"`). On a alors toute la puissance de SQL (pour faire une simple table de hachage, ça peut être beaucoup).

Pour comparer trois de ces bibliothèques, je suis parti d'un programme qui analyse des requêtes faites à un serveur DNS faisant autorité. On veut savoir quels domaines déclenchent le plus souvent une réponse NXDOMAIN (indiquant que le domaine n'existe pas). Pour cela, on ouvre le fichier au format pcap, on analyse le contenu (<http://www.bortzmeyer.org/libpcap-c.html>) et, pour chaque réponse NXDOMAIN, on regarde la table de hachage. Si le domaine y est déjà, on incrémente le compteur. Sinon, on crée une entrée avec un compteur initialisé à 1. À la fin, on affiche le résultat. Trivial avec un langage comme Perl, voici le programme en C (en ligne sur <http://www.bortzmeyer.org/files/test-hashtable-nxdomain.c>) (et son fichier d'en-tête (en ligne sur <http://www.bortzmeyer.org/files/test-hashtable-nxdomain.h>)). Pour le compiler, le plus simple est d'utiliser un Makefile du genre de :

```
#HASHTABLE=HASHTABLE_JUDY
#HASHTABLE=HASHTABLE_GLIB
HASHTABLE=HASHTABLE_HAMSTERDB

ifeq (${HASHTABLE}, HASHTABLE_GLIB)
HASHTABLE_CFLAGS=$(shell pkg-config --cflags glib-2.0)
else ifeq (${HASHTABLE}, HASHTABLE_JUDY)
HASHTABLE_CFLAGS=
else ifeq (${HASHTABLE}, HASHTABLE_HAMSTERDB)
HASHTABLE_CFLAGS=
else
HASHTABLE_CFLAGS=
endif

ifeq (${HASHTABLE}, HASHTABLE_GLIB)
HASHTABLE_LDFLAGS=$(shell pkg-config --libs glib-2.0)
else ifeq (${HASHTABLE}, HASHTABLE_JUDY)
HASHTABLE_LDFLAGS=-lJudy
else ifeq (${HASHTABLE}, HASHTABLE_HAMSTERDB)
HASHTABLE_LDFLAGS=-lhamsterdb
else
HASHTABLE_LDFLAGS=
endif

CFLAGS=-D${HASHTABLE} ${DEBUG} -Wall ${HASHTABLE_CFLAGS}
LDFLAGS=-lpcap ${HASHTABLE_LDFLAGS}
...
```

En faisant varier HASHTABLE, on peut tester le programme avec quatre bibliothèques différentes et vérifier que tout marche bien.

Et leurs performances ? Avec un fichier pcap de 55 millions de paquets DNS (7,6 Go), dont 3,7 millions renvoient NXDOMAIN, ce qui fait 310 000 domaines à mettre dans la table de hachage, sur un PC muni d'un Pentium à 2,66 Ghz et de 2 Go de RAM, avec Debian, HamsterDB l'emporte haut la main en 4 minutes et 10 secondes. GLib est loin derrière avec 14 minutes et 55 secondes.

Merci à Kim-Minh Kaplan pour le support de BerkeleyDB dans le programme.

Attention, il s'agit de comparaison de mes programmes, pas de ces bibliothèques. Pour un autre problème, codé différemment, le « match » aurait pu donner des résultats bien différents.