Systèmes d'exploitation – 2015/2016 Printemps

Rapport projet – Luigi Coniglio

Performances des fonctions de hachage

Vous trouverez dans le fichier kv.c trois fonctions de hachage différentes, respectivement: hash_fun1, hash_fun2 et hash_fun3. Ces trois fonctions renvoient toujours une valeur comprise entre 0 et 999982.

La première fonction de hachage correspond à celle décrite dans les spécifications du projet (algorithme loselose). La deuxième effectue une compression de la clef en utilisant l'opération xor, en plus elle réalise des décalages des bits de chaque caractère de la clef (sur la longueur d'un len_t) pour contourner des collisions en cas de permutations de la clef (ex: «pablo picasso» n'aura plus à partager son bloc dans le fichier .blk avec «pascal obispo»).

La troisième fonction de hachage correspond (sauf le modulo) à la fonction de Fowler–Noll–Vo. Cette fonction effectue une compression de la clef en utilisant l'opération xor et des multiplications par un nombre premier pour assurer une bonne distribution des valeurs.

Mesures de performance des fonctionnes de hachage

Dans les fichiers rendu vous trouverez un programme hash_gen.c et un script count_collisions.sh . Le premier programme génère un nombre à votre choix de hash

(10 par défaut, sinon utiliser l'option -s) sur des clef aléatoires en utilisant une des fonctions de hachage disponibles (option -i).

Le script count_collisions.sh permet de calculer le nombre de collisions générée par chaque fonction de hachage, pour l'utiliser vous devez d'abord avoir compilé le programme hash_gen.

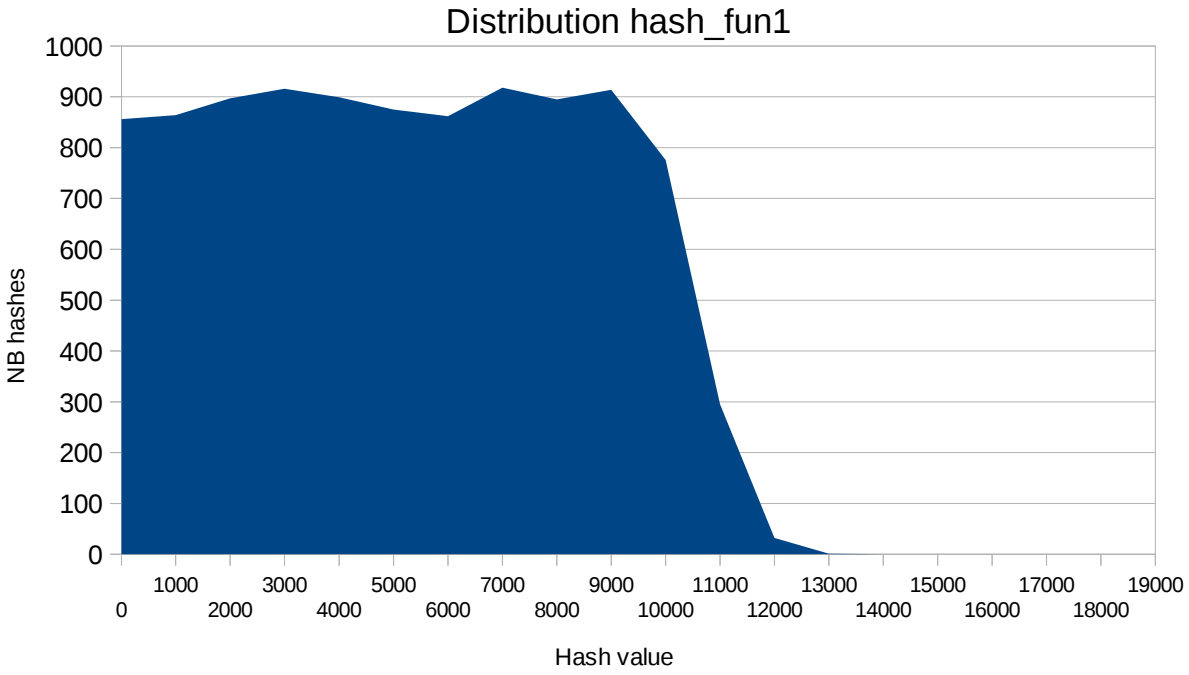
Distribution et collisions

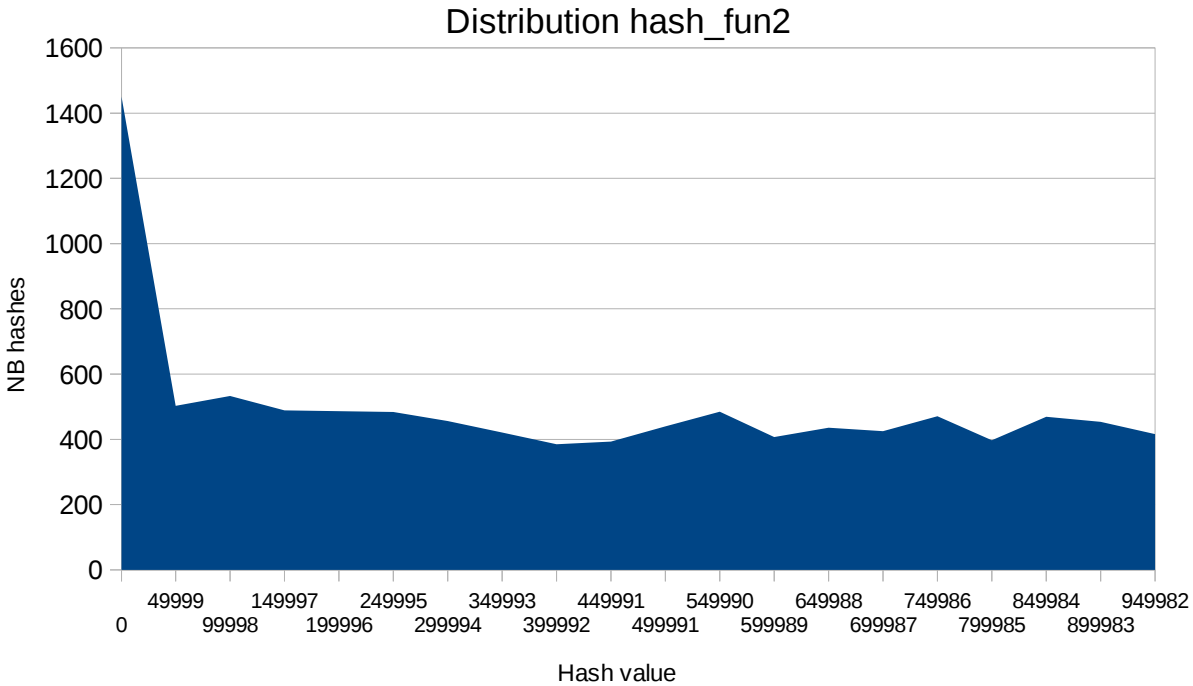
Pour mesurer la distribution des valeurs données par chaque fonction, j'ai généré 10000 haches sur des clefs aléatoires avec une longueur maximale de 100 caractères

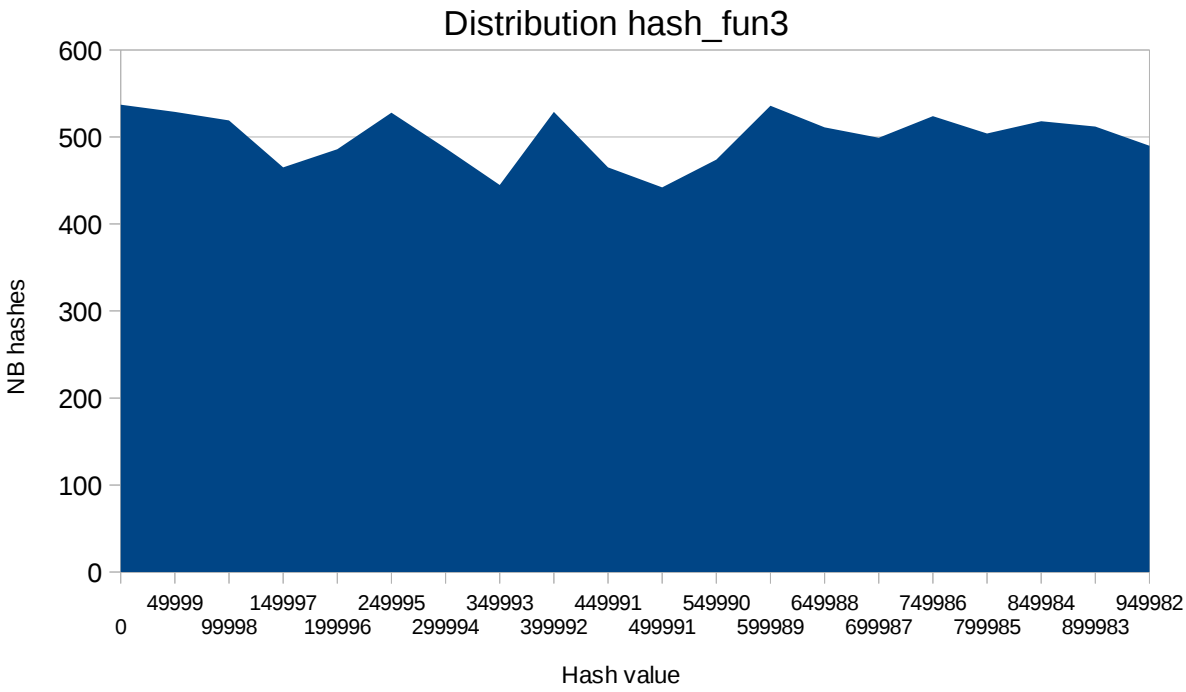
et j'ai créé des histogrammes représentent la fréquence avec laquelle les valeurs apparaissent

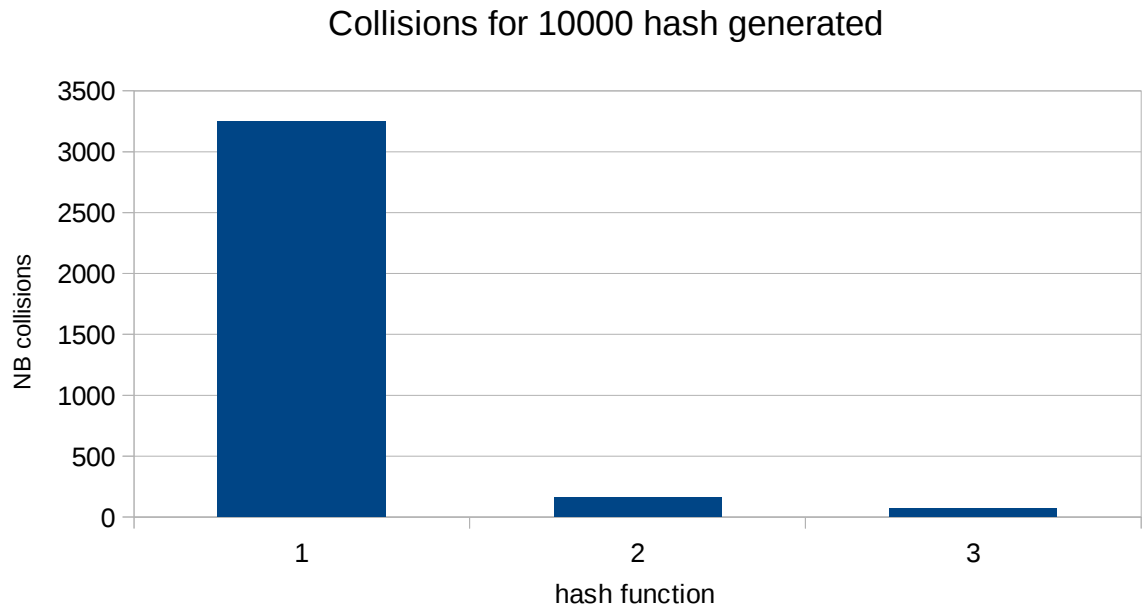
La première fonction de hachage tend à générer des valeurs très petites (dans ce test les valeurs sont toujours plus petites de 13000) en fait dans cette fonction la valeur renvoie dépend directement de la longueur de la clef (généralement plus la clef est longue plus le hash généré est grand). En pratique pour être sûr de pouvoir atteindre tous les valeurs comprises entre 0 et 999982 on devrait utiliser des clefs avec une longueur maximale de 3937 caractères! A cause de cela cette fonction est aussi la moins performante du point de vue des collisions.

La deuxième et la troisième fonctions de hachage offrent des performances beaucoup plus élevées que la première. La troisième fonction de hachage, particulièrement, est la plus uniforme.



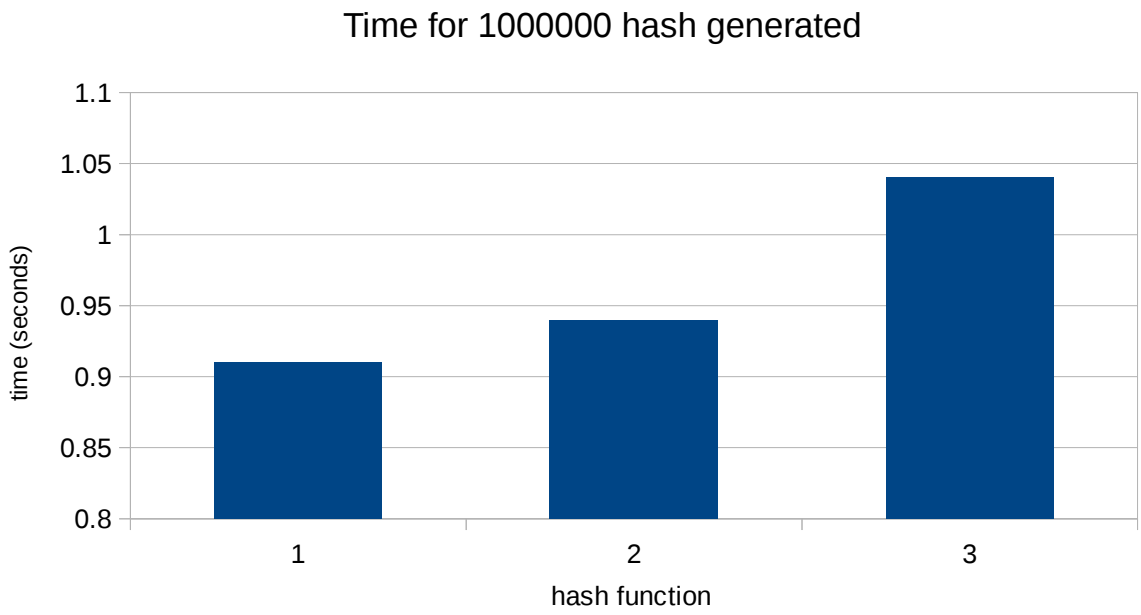






Vitesse

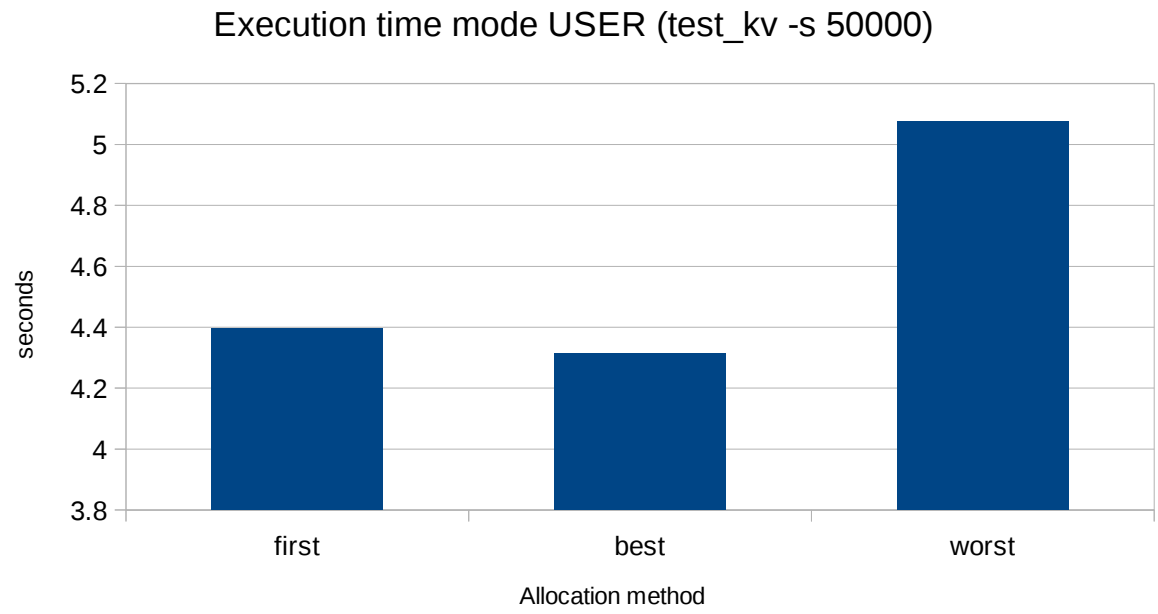
La première fonction est légèrement plus rapide par rapport aux autres fonctions, mais cette différence de quelque dizaine de nanosecondes peut être considéré négligeable par rapport au gain de homogénéité dans la distribution des hashes et la diminution des collisions.

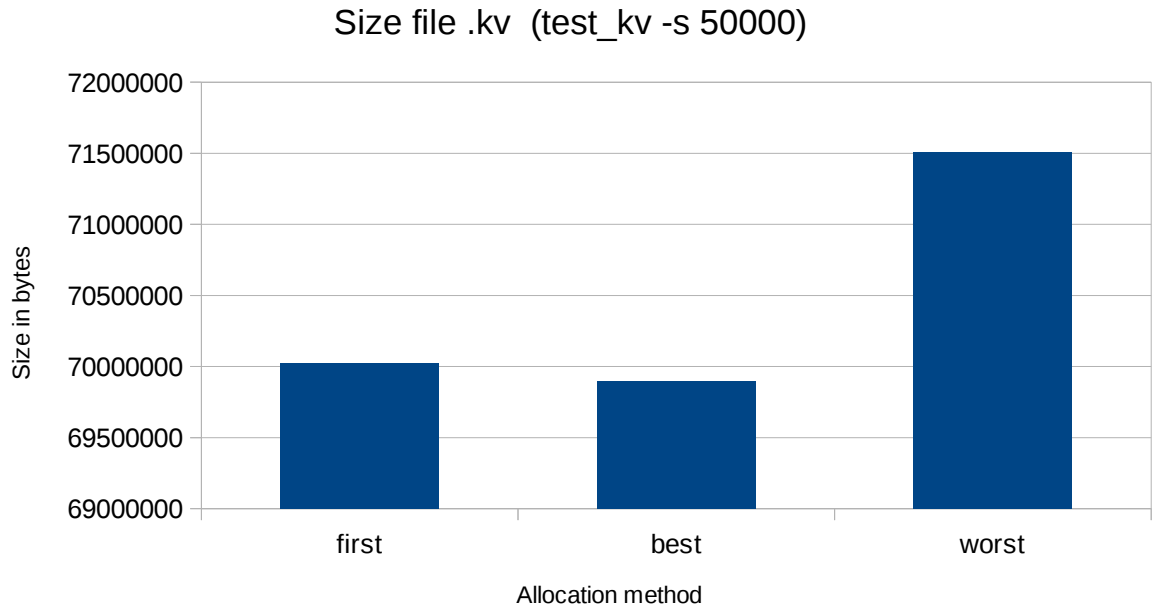


Mesures de performance des méthodes d'allocation

Le programme test_kv.c génère une base de données et effectue un certain nombre de kv_put avec des clef aléatoires dont, pendant l’exécution, certaines sont supprimées avec kv_del pour créer des trous. Les scripts time_test.sh et size_test.sh utilisent test_kv.c pour mesurer la durée d’exécution et la taille de la base de données crée.

On peut voir que les résultants des deux test (test sur le temps et test sur la taille) sont très proches. La méthode d'allocation "best fit" est la meilleure, la méthode "first fit" est très proche d’être la meilleure méthode et la méthode "worst fit" est sans doute la moins performante (spécialement du point de vue du temps: environs 25% plus lente que "best fit").


Execution time mode USER (test_kv -s 50000)
5.2
5
4.8
4.6
4.4
4.2
4
3.8
seconds
first
best
worst
Allocation method


Size file .kv (test_kv -s 50000)
72000000
71500000
71000000
70500000
70000000
69500000
69000000
Size in bytes
first
best
worst
Allocation method