TP2_ARSE

October 16, 2022

1 Rappel-

2 Analyse d'un benchmark

2.1

On retrouve l'algorithme dans le fichier bench.c, fonction benchRead lignes 111 à 142.

2.2

Un benchmark est un test permettant de mesurer les performances d'un système ou d'un programme pour le comparer à d'autres.

2.3

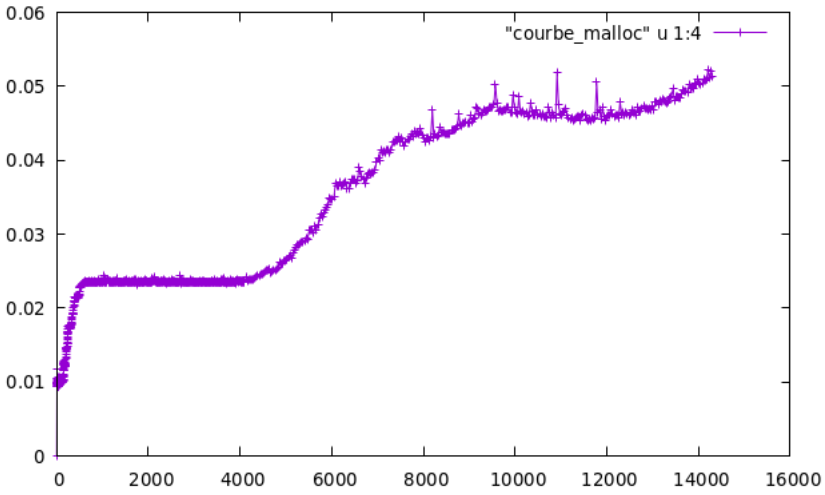
On préchauffe le cache en faisant quelques boucles afin de ne pas subir la lenteur de l'initialisation de la RAM qui peut engendrer des résultats aléatoires (sinon il faudrait faire une médiane pour aplanir les résultats).

2.4

On effectue:

```
./bench_malloc > courbe_malloc
gnuplot
plot "courbe_malloc" u 1:4 w lp
```

Temps d'exécution en fonction de la taille du buffer:



2.5

La courbe a 2 paliers, l'utilisation du cache ne semble pas optimale.

2.6

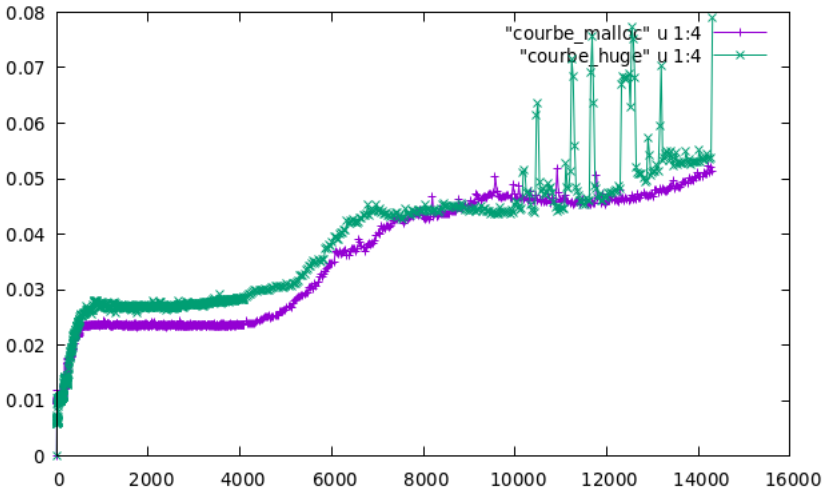
Les huge pages sont des pages mémoires de taille bien supérieure au pages classiques de 4Ko. La zone mémoire de ces pages est bloquée en RAM et ne peut pas aller en SWAP et permet d'optimiser la gestion du TLB. Ces pages ont donc un impact significatif sur l'utilisation de la CPU lors des accès mémoire.

2.7

Dans la machine virtuelle (pour les droits de création du dossier + montage de la huge_page)

```
//echo 200 > /proc/sys/vm/nr_hugepages
//mkdir /mnt/huge
//mount -t hugetlbfs -o mode=0777 none /mnt/huge
```

Temps d'exécution en fonction de la taille du buffer:



On obtient un success quand la page a pu être allouée.

2.8

On constate que les huge_pages s'éxécutent un peu plus rapidement que les pages classiques, toutefois pour des tailles plus grandes le temps d'éxécution est beaucoup plus variable, ponctuellement 2 fois plus rapide mais quelques fois moins rapide.

Bonus

2.9

La mémoire des huge_pages étant bloquée en RAM, si une trop grande quantité de mémoire est allouée il se peut, le temps d'éxécution étant assez instable pour ces quantités, que le système plante.

3 Création d'un allocateur en mode utilisateur

3.1 10 (malloc dans hp_allocator_malloc/SRC/)

hp_init alloue des huge pages (ouverture d'un fichier en écriture et lecture que l'on mappe en mémoire) et hp_finalize les désalloue.
Il y a initialisation au moment du chargement de la librairie dynamique.

3.2 11

alloue représente la taille de la mémoire à allouer.

3.3 12

On déclare alloue en static car sa taille est fixe (huge_page).

3.4 13

On implémente la fonction __hp_malloc en s'assurant de disposer d'assez de mémoire et de réserver la mémoire demandée (hp_allocator.c lignes 39 à 62):

```
if (alloue+taille>MEM_SIZE)
{
    fprintf(stderr, "Il n'y a plus assez de m moire disponible.\n");
    return NULL;
}

ret = alloue+&reserve.mem;
alloue += taille;
return ret;
```

3.5 14

Les blocs alloués pour ces pages ne peuvent pas être libérés et occupent une taille toujours plus grande en mémoire (pas de limite quant aux ressources disponibles).

3.6 15 (free dans /hp_allocator_malloc_free/SRC/)

recherche_bloc_libre regarde les blocs libres allouables ayant une taille suffisante dans une page

3.7 16

(hp_allocator.c lignes 83 à 97)

```
while (bloc->suivant!=NULL/*??DONE??*/)
    {
        if (bloc->taille >= taille)
        {
            if (precedent != NULL)
                precedent->suivant = bloc->suivant;
            else
                libre = bloc->suivant;

            ret = bloc;
            break;
        }
```

3.8 17

__hp_malloc__ permet d'allouer dans ret des blocs de taille taille tout en limitant la taille alloué par rapport aux ressources disponibles avec la recherche des blocs libres.

3.9 18

Comme pour la question 13, en prenant en compte la nouvelle gestion des blocs (ligne 121):

```
ret = *recherche_bloc_libre(taille); /*??DONE??*/ //recherche d'un bloc libre
```

3.10 19

__hp_free__ libère les blocs alloués par __hp_malloc__

3.11 20

4 Bibliothèque dynamique

4.1 22

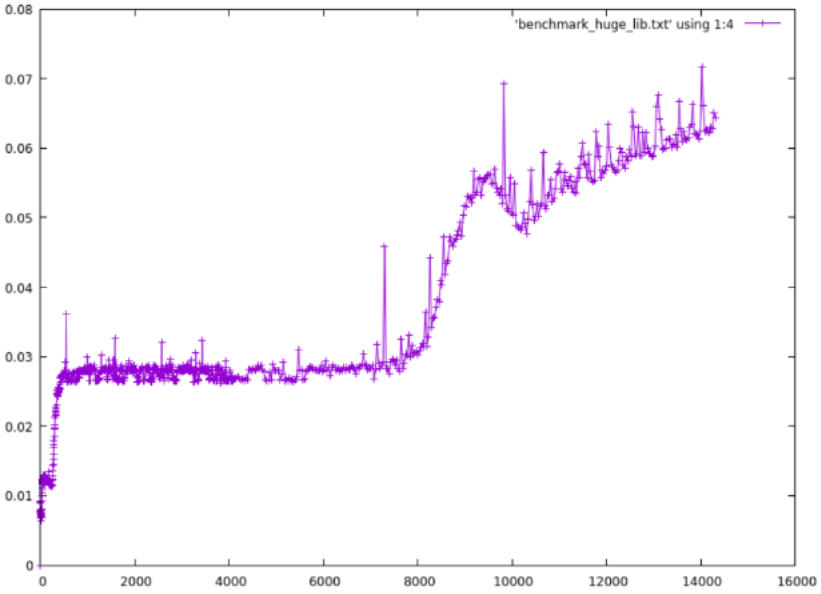
Il existe une variable d'environnement: LD_PRELOAD qui sert à charger une librairie avant d'éxecuter un programme, par exemple :

LD_PRELOAD=/lib64/mylibc.so /bin/foo éxecute mylibc.so avant /bin/foo.

On peut donc effectuer:

```
gcc -o libhp_allocator.so -fpic -shared libhp_allocator.c
#a partir de la librairie disponible
export LD_LIBRARY_PATH=../hp_allocator/hp_allocator_lib/lib/
make
ldd ./bench_hp_malloc
```

En utilisant la bibliothèque fournie, on obtient les résultats suivants :



On remarque que le temps d'accès décolle 2ko plus loin qu'avant.