Architecture d'un système d'exploitation

(ASE 2022-2023)

TD2

Interaction de la pagination avec les caches

hugo.taboada@cea.fr
marc.perache@cea.fr

Click here or scan to download TD'files from pcloud link.

Le but de ce TD est de comprendre le fonctionnement des caches et leur interaction avec la pagination. Dans un premier temps, nous analyserons un benchmark permettant d'observer ces interactions. Nous nous appuierons sur les travaux de Sébastien Valat tutoré par M. Marc Pérache. La thèse est disponible à l'adresse suivante : `https://tel.archives-ouvertes.fr/tel-01253537/document`. Dans un second temps, nous vous proposons de créer un allocateur en mode utilisateur. Le but de cet allocateur est d'allouer de la mémoire de manière contigüe en mémoire physique.

I Rappel : Interaction de la pagination avec les caches

Les modes associatifs des caches posent un problème de collision de lignes de cache. En effet dans ce fonctionnement, chaque adresse est associée à une ou plusieurs lignes de cache. Si à un instant donné le processus courant essaie d'accéder à des informations associées aux mêmes lignes de cache ; alors le processeur passera son temps à vider et remplir ses lignes de cache ne profitant pas de la taille complète du cache. On rappelle qu'à cause de la pagination, des données contiguës en adresses virtuelles ne le sont pas en mémoire physique comme cela est illustré par la figure 1. Sous Linux les pages sont en effet distribuées "aléatoirement" aux applications. Il est donc probable qu'une application obtienne successivement deux pages physiques dont l'alignement correspond aux mêmes emplacements dans les voies du cache. Si l'application reçoit plus de pages de ce type qu'il n'y a de voies dans le cache, et qu'elle accèdent régulièrement à l'ensemble de ces données, il y aura collision, et l'application sera ralentie même si toute sa mémoire pourrait rentrer dans le cache. Ce cas est illustré par la figure 1.

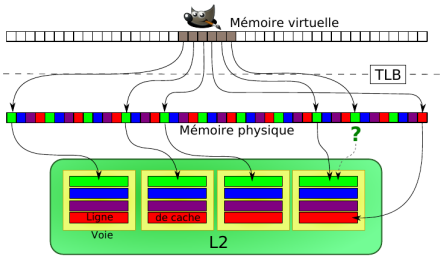

FIGURE 1 – Exemple de conflits de ligne de cache

II Analyse d'un benchmark

Nous vous proposons d'analyser un benchmark dans le but de comprendre le fonctionnement des caches et leur interaction avec la pagination.

Data: ENTIER : taille
Data: DOUBLE : buffer[taille]
Data: ENTIER : i, j
PRE_CHAUFFE_DU_CACHE(buffer,taille);
LANCER_CHRONO();
for $j \leftarrow 0$ **to** 640000000/*taille par pas de 1* **do**
  for $i \leftarrow 0$ **to** *taille par pas de 16* **do**
    LIRE(buffer[i]);
  end
end
ARRETER_CHRONO();

Algorithm 1: Pseudo code de test des caches.

Q.1: Dans quelle fonction est codé cet algorithme ? (nom du fichier + ligne)

Q.2: Expliquez le fonctionnement du benchmark. Quel est son but ?

Q.3: Pourquoi préchauffe-t-on le cache ?

Q.4: Lancez le benchmark *malloc_bench* puis tracez une courbe représentant le temps d'exécution en fonction de la taille du buffer.

Q.5: Que remarquez-vous ? L'utilisation du cache est-elle optimale ? Expliquez l'allure de la courbe obtenue.

Q.6: L'utilisation du benchmark *huge_bench* requiert l'utilisation des huges pages. Qu'est-ce qu'une huge page ?

Q.7: Après avoir activées les huges pages, lancez le benchmark *huge_bench* puis tracez une courbe représentant le temps d'exécution en fonction de la taille du buffer.

Q.8: Que remarquez-vous ? L'utilisation du cache est-elle optimale ? Expliquez l'allure de la courbe obtenue.

Q.9: Bonus : Quels problèmes peut-on rencontrer avec l'utilisation des huges pages ?

III Création d'un allocateur en mode utilisateur

Nous vous proposons maintenant de créer un allocateur en mode utilisateur. Le but de cet allocateur est d'allouer de la mémoire de manière contigüe en mémoire physique. Pour cela, nous utiliserons les huges pages.

1 Gestion du malloc

Dans le dossier **hp_allocator/hp_allocator_malloc/SRC/** :

Q.10: Dans le fichier *hp_allocator.c*, que font les fonctions **hp_init** et **hp_finalize** ? À quel moment ce code est-il appelé ?

Q.11: À quoi sert la variable **alloue** ? Pourquoi est-elle déclarée en static ?

Q.12: Expliquez le fonctionnement de la fonction **___hp_malloc**.

Q.13: Implémentez la fonction **___hp_malloc**.

Q.14: Quel est le problème de cette implémentation ?

2 Gestion du free

Dans le dossier **/hp_allocator_malloc_free/SRC/** :

Q.15: Expliquez le fonctionnement de la fonction **recherche_bloc_libre**.

Q.16: Implémentez la fonction **recherche_bloc_libre**.

Q.17: Expliquez le fonctionnement de la fonction **___hp_malloc**.

Q.18: Implémentez la fonction **___hp_malloc**.

Q.19: Expliquez le fonctionnement de la fonction **___hp_free**.

Q.20: Implémentez la fonction **___hp_free**.

IV Bibliothèque dynamique

Q.21: Faire une bibliothèque dynamique de votre allocateur.

Q.22: Linkez votre bibliothèque dynamique avec le code dans le dossier *svalat_bench_lib* puis tracez une courbe. Que remarquez-vous ? (Si vous n'avez pas réussi à faire votre propre bibliothèque dynamique, vous pouvez utiliser celle dans *hp_allocator/hp_allocator_lib*)

V Wrapper la fonction malloc

Q.23: Wrapper les fonctions **malloc** et **free** pour utiliser votre allocateur à la place du malloc sans toucher au code du benchmark dans le dossier *svalat_bench*

Ce TD est noté. Le code source et le rapport sont à fournir. Attention à bien faire des Makefile. Merci de bien préciser dans le rapport à quelle ligne, et dans quel fichier, se trouvent vos réponses dans le code source. Détaillez l'implémentation dans le rapport. Tout cela est à envoyer avant le 16/10/2022 à 23h59 sur le lien pcloud suivant :

https://e.pcloud.com/#page=puplink&code=yFeZCA43F45wWFyreyfxYNxxoHBF1r07

Pour toutes questions n'hésitez pas à m'envoyer un mail à l'adresse suivante : *hugo.taboada@cea.fr*