15 mai 2014

LSINF1252
SYSTEMES INFORMATIQUES 1

Rapport du Projet 3

Ndizera Eddy
Solaiman El Jilali

Introduction

Dans le cadre de ce projet, il nous a été demandé d'implémenter 2 benchmarks : un portant sur l'appel système readdir et un autre afin de comparer writev avec lseek+write. Ce rapport contiendra des explications sur nos choix de scénarios et d'implémentation, une analyse des résultats de nos benchmarks avec graphe à l'appui ainsi que les difficultés rencontrées.

1 Choix de scénarios et d'implémentation

1.1 writev/lseek+write

Le scénario considéré pour ce benchmark s'est basé sur 2 critères : la taille du buffer (iov pour writev) ainsi que la taille des fichiers à écrire. La taille des buffer nous a été inspiré par le fait que writev écrit par blocs (iovec) dont la longueur peut être fixé tandis que write écrit par buffers dont la longueur aussi peut être fixé. De plus, comme on calcule le temps moyen par appel système(temps pour écrire le fichier divisé par le nombre d'appels à la méthode), cela nous permet de savoir lequel des deux est le plus rapide pour une taille de bloc définie. Le critère de taille des fichiers a aussi son importance dans le sens qu'il permet de comparer les performances des 2 appels sur une durée plus longue.

Pour l'implémentation du benchmark, nous avons écrit 2 méthodes *benchmark_writev* et *benchmark_lseek* qui prennent en argument un descripteur de fichier (fd), une taille du buffer/iov (buffer_size), une taille de fichier à écrire (file_size), un timer (t) et en dernier lieu un recorder (rec). Ces 2 méthodes, comme leur nom l'indique, sont utilisées pour calculer le temps que prend chaque appel système. ? ? ?

benchmark_writev crée et initialise un tableau de iovec qui sont les blocs à écrire. Une fois l'initialisation terminée, il démarre le timer et fait appel à writev qui va se charger d'écrire les blocs dans un fichier. L'écriture étant terminé, il stoppe le timer et enregistre le temps moyen pris pour l'écriture d'un bloc.

benchmark_lseek fonctionne de manière similaire sauf qu'il va démarrer le timer et entrer dans une boucle qui prendra fin quand il aura écrit un nombre de données égal à la taille du fichier à écrire. A l'intérieur de la boucle, la méthode écrit une chaîne de caractères dans un fichier via write puis repositionne le curseur du ficher à la fin de celui-ci. Lorsqu'on sort de la boucle, on arrête le timer et on enregistre le temps moyen pris pour écrire un bloc (ou buffer).

La méthode main se chargera simplement de faire appel à ces 2 méthodes en changeant leurs paramètres. Ce choix d'implémentation permet une modularité dans le sens où on peut facilement changer les tailles des buffer ou des fichiers à tester ou même en ajouter.

Nous tenons également à souligner que la méthode lseek n'est pas indispensable puisque write écrit à partir de la fin du fichier et donc, qu'il n'est pas nécessaire de repositionner le curseur du fichier chaque fois à la fin.

1.2 readdir

Le scénario considéré pour pour ce benchmark se base sur le nombre de fichiers contenus dans le répertoire sur lequel l'appel système readdir s'appliquera.

En effet, intuitivement, nous pensions que le temps mis pour lister les fichiers contenu dans un dossier serait proportionnel au nombre de fichiers qu'il contient. Pour vérifier cela, nous avons implémenté un benchmark générant un grand nombre de fichiers et lisant à intervalle régulier le répertoire. Pour travailler proprement, une partie de la fonction main consiste à créer un sous-répertoire ./temp au répertoire courant et c'est dans ce répertoire que seront créés l'ensemble des fichiers. Soulignons que ce répertoire sera supprimé après l'application du benchmark afin de ne pas polluer inutilement le dossier de benchmark.

Dans l'état actuel de notre benchmark, nous générons 10000 fichiers dans le dossier ./temp et tous les 100 fichiers (à partir de 1 fichier), nous lançons *benchmark_readdir*. Lors de l'exécution de *benchmark_readdir*, nous démarrons un timer juste avant le parcourt complet du répertoire ./temp et l'arrêtons une fois que tous les fichiers de ce répertoire ont été lus. Dans la boucle parcourant ce répertoire nous incrémentons un compteur qui n'est effectivement pas compris dans l'appel système readdir. Cela dit, nous avons supposé cette incrémentation négligeable. De plus, cette opération s'effectue à chaque itération de la boucle. De ce fait, le "coût" de cette incrémentation est uniformément répartie sur chacune des lecture. De plus, nous tenons à souligner que le répertoire crée ainsi que l'ensemble des fichiers créés sont supprimés après l'application de notre benchmark.

2 Analyse des résultats

2.1 writev/lseek+write

En se référant a la figure 1, on constate que pour une taille de fichier plus petite, lseek+write prend moins de temps en moyenne pour écrire un bloc que writev. Cependant, pour un plus gros fichier, lseek+write prend cette fois-ci plus de temps que writev. Cette différence peut peut-être s'expliquer par l'appel à lseek qui devient plus coûteux au fur et à mesure que le fichier grossit.(POURQUOI)

De plus on constate que pour writev, le temps moyen par écriture est moindre pour un fichier de grande taille que pour un fichier de petite taille. Au contraire de lseek+write où le temps moyen est supérieure pour un fichier de grande taille. Une dernière observation est que le temps pris pour écrire augmente avec la taille des blocs à écrire, ce qui est tout à fait normale.

2.2 readdir

Pour la représentation graphique de ce benchmark, nous avons choisis des échelles linéaires. En se référant aux différentes figures générées, on constate que le temps mis pour parcourir complètement un répertoire est linéaire et proportionnel au nombre de fichiers contenus dans se ce répertoire. Quelques lectures de répertoire prennent légèrement plus de temps que d'autres. Cela est probablement dû au fait que le processus est mis en attente lorsqu'un autre processus fait des lectures.

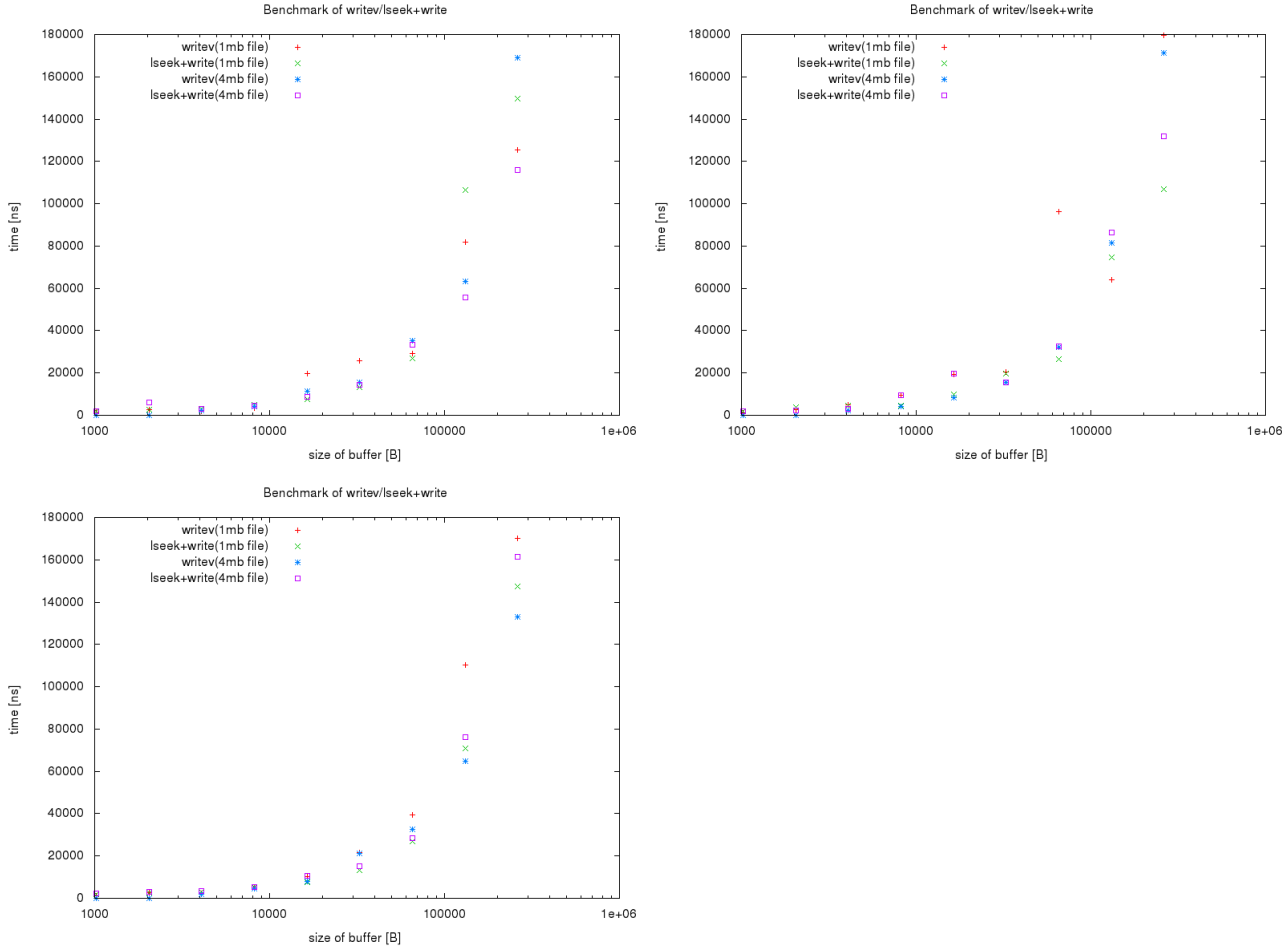

FIGURE 1 – Graphe writev/lseek+write en fonction du temps et de la taille des buffer/iov

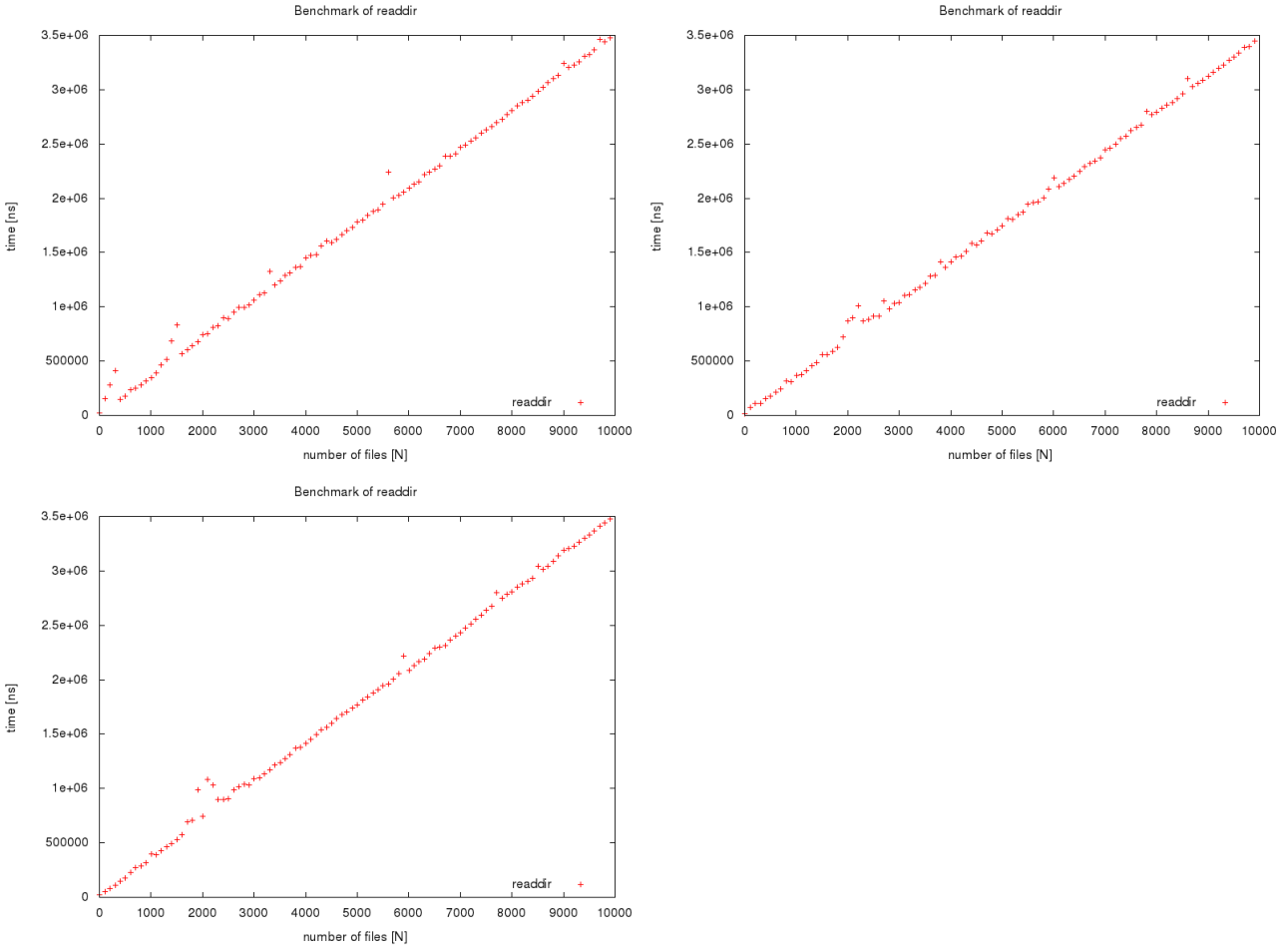

FIGURE 2 – Graphe du temps mis par l'appel systeme readdir sur un répertoire en fonction du nombre de fichier dans ce repertoire

3 Difficultés rencontrées

Une difficulté rencontrée fut de savoir quels fichiers modifiées pour prendre en compte nos benchmarks. En effet, nous avons du chercher à plusieurs endroits pour savoir quels fichiers ou lignes ajouter pour que le projet prenne bien en compte nos modifications (creation du make automatique pour nos projets). A la génération des fichiers dans le benchmark de readdir, nous avons eu quelques difficultés liées au fait que nous ne rendions pas compte qu'il fallait ouvrir et fermer le répertoire à chaque appel du benchmark pour que la lecture se fasse effectivement depuis le début du répertoire.

Conclusion