16 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 un critère : la taille du buffer (iov pour writev). La taille des buffer nous a été inspiré par le fait que *writev* écrit par bloc (iovec) dont la longueur peut être fixé tandis que *write* écrit par buffers dont la longueur aussi peut être fixée. Pour ce scénario, on compare le temps pris par *writev* et *lseek+write* pour écrire un fichier d'une taille fixe mais pour des tailles de buffers différentes. D'autres scénarios ont été considérés comme le fait de d'utiliser 2 tailles de fichiers mais ils n'ont pas été repris du fait du peu d'informations qu'ils ajoutaient en plus.

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'indiquent, sont utilisées pour calculer le temps que prend chaque appel système writev et lseek+write.

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 terminée, il stoppe le timer et enregistre le temps pris pour l'écriture d'un fichier.

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 une quantité de données égale 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 pris pour écrire un fichier.

La fonction 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.

Nous tenons également à souligner que la fonction *lseek* n'est pas indispensable puisque write écrit à partir de la fin du fichier et donc, 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 du premier 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.

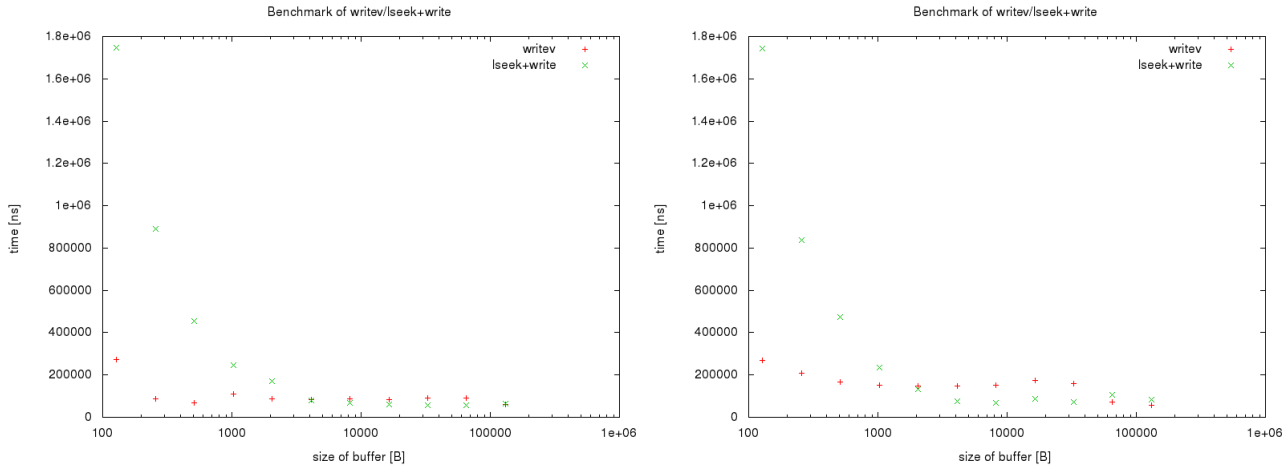

FIGURE 1 – Graphe writev en fonction du temps et de la taille des buffer/iov

D'autre part, le premier benchmark que nous avons élaboré se basait sur la taille des fichiers à parcourir par *readdir*. Nous nous sommes rapidement rendu compte que l'appel système *readdir* est indépendant de la taille des fichiers à parcourir. De ce fait, ce benchmark n'a pas été retenu.
Subséquemment, étant donné qu'un répertoire est traité comme un fichier dont le contenu est la liste des fichiers référencés [1], il n'y a donc aucun intérêt à générer des répertoires à coté des fichiers générés.

2 Analyse des résultats

2.1 *writev/lseek+write*

En se référant à la figure 1, on constate que le temps pris pour écrire un fichier d'une taille fixe par *lseek+write* décroît quand le buffer augmente jusqu'à se stabiliser à partir d'une certaine taille de buffer. On observe de plus que l'appel à writev est en moyenne plus rapide pour écrire que *write+lseek*. Ceci est d'autant plus visible que le buffer est petit. ? ? ? ? ? ?Cela s'explique par le fait que plus le buffer est petit plus grand le nombre d'appels systèmes se font. Comme *lseek+write* fait appel à 2 appels systèmes que sont *write* et *lseek*, plus on fait appels à eux plus il est normal qu'ils prennent plus de temps même si au final ils écrivent la même quantité d'informations. Une dernière observation est le fait que *writev* garde un temps constant quelque soit le buffer.

En conclusion, il est plus intéressant d'utiliser *writev* que *lseek+write*. L'appel système *writev* est plus rapide en moyenne que *lseek+write* et a l'avantage de garder un temps constant pour une même taille de fichier.

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.

3 Difficultés rencontrées

Une difficulté rencontrée fut de savoir quels fichiers modifier 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

1. Voir page 145 du syllabus SINF1252-Theorie

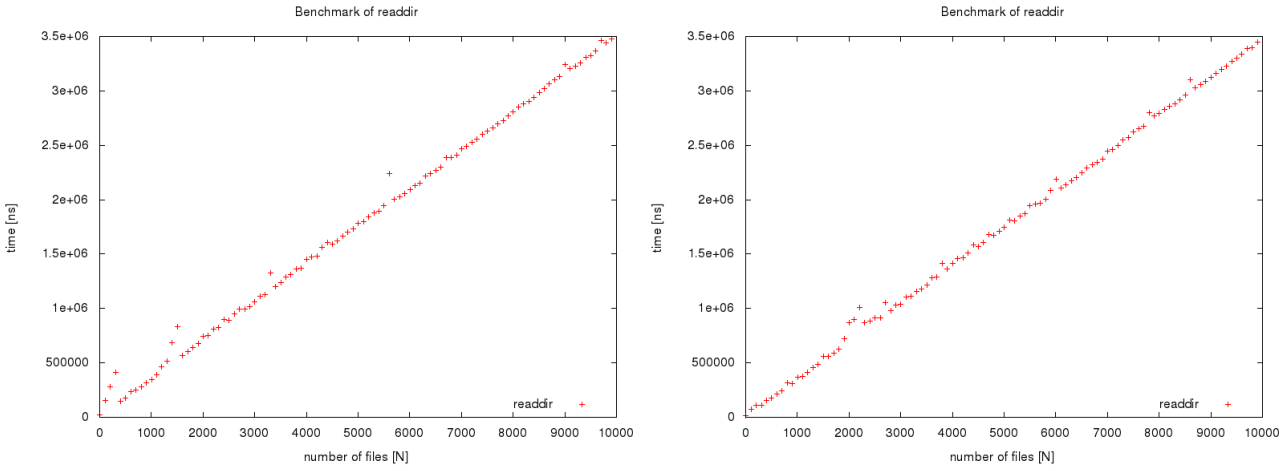

FIGURE 2 – Graphe du temps mis par l'appel systeme readdir sur un répertoire en fonction du nombre de fichier dans ce repertoire

en compte nos modifications (creation du make automatique pour nos projets).

Une autre difficulté fut de savoir quels critères choisir pour son benchmark et notamment faire un choix entre temps moyen et temps total. En effet, les résultats du benchmark peuvent se voir changer et on peut en arriver à d'autres conclusions. C'est ce qui nous est arrivé avec notre *benchmark* pour lseek+write.

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.