Sorbonne Université

Rapport Projet SAR 2024

Éditeur de Fichiers Coopératif et Réparti

TO Jérémy 21314557

Bewa Alexandre 21312673

Mai 2024

Table des matières

Chapitre 1

Introduction

1.1 Contexte

Dans le cadre du projet de Master SAR de second semestre nous avions comme objectif de créer un éditeur de fichier coopératif.

L'intérêt est d'utiliser et consolider des connaissances établies aux UEs et semestres précedents.

1.2 Problématique

La difficulté principale repose sur l'utilisation des variables et de garder leurs cohérences entre plusieurs threads. Pour cela une approche proposée est l'utilisation de mutex et de verrous pour rendre les opérations sur les variables atomiques et les protéger des modifications d'autres threads.

1.3 Implémentation

Nous avons décidé d'utiliser **Java** comme langage de programmation pour ce projet, dû à la facilité d'utilisation des objets, à la grande quantité de de documention disponible ainsi que notre propre affinité avec ce langage.

Chapitre 2

Structures de données

2.1 Explications

Le Fichier est stocké en plusieurs versions : Une principale par le Serveur et une locale par chaque client connecté.

La difficulté est de garantir la cohérence entre toutes ces versions d'un même fichier. Nous avons donc décidé d'utiliser la politique de cohérence : **UPDATE**.

Notre fichier coopératif contient deux structures de données principales :
- Fichier
- Ligne

2.2 Fichier

La structure *Fichier* est une conjonction :
- entier global : *Compteur*
- entier global : *IdCourant*
- *Liste* de *Ligne* : *ensemble*
- *Map* : *Locks* ; pouvant contenir une paire d'un identifiant de Ligne et d'un identifiant d'utilisateur.

IdCourant

Cette variable globale augmente à chaque création de Ligne. Ne décrémente jamais. Permet de générer de nouveaux identifiant.

Compteur

Cette variable globale augmente à chaque création de Ligne. Décrémente à chaque suppression.

Map : Locks

L'intérêt de la Map *Locks* est de permettre et d'empécher les modifications de Lignes simultanées. Lorsqu'une Ligne l est entrain d'être modifiée par un utilisateur u la paire $<l,u>$ est stockée dans *Locks*. Si un autre utilisateur demande une action de modification ou de Supression sur la Ligne, l'action ne sera refusé par le serveur. À la fin de la modification de la Ligne l, la pair est enlevé de la Map.

2.3 Ligne

La structure *Ligne* contient :

- un entier unique : $identifiant$
- une chaîne de charactère : *contenu*
- un Bouléen *lock*

Identifiant

L'$identifiant$ de la ligne est unique. Seul le Serveur gère la création d'identifiant de Ligne à l'aide d'une variable globale garantissant ainsi leur unicité.

lock

Le bouléen *lock* est une seconde mesure de sécurité posé dans le cas de modification ou de suppression de Ligne.

Chapitre 3

Requêtes

3.1 Modification de Lignes

La modification de Ligne se fait en 4 étapes :
- Client requête un modification de Ligne
- Serveur Acquièce ou refuse
- Client envoie la ligne modifiée au Serveur
- Serveur enregistre la ligne et envoie à tous les client la nouvelle ligne

3.2 Création et Suppression de Lignes

Les requêtes de Création et Suppression de Ligne sont prise en compte par le Serveur l'une après l'autre.

Creation :
- Client demande de créé une ligne après une ligne spécifiée $[-1, N]$
- Le serveur crée la Ligne avec un identifiant unique et envoie cette ligne à tous les clients

Suppression :
- Client demande la suppression d'une ligne
- Le serveur verifie si la ligne est bloquée
- Si elle ne l'est pas supprime la ligne et envoie un message à tous les clients

3.3 Synchronisation

Cette requête se fait à chaque connexion d'un client et sur volonté du client
- Client demande de se synchroniser avec le serveur
- Serveur envoie le contenu du fichier au Client

3.4 Sauvegarde

- Client demande la sauvegarde du fichier au serveur
- Serveur sauvegarde le fichier si aucune requête de Modification est pendante

Chapitre 4

Algorithmes

Algorithm 1 Modification de Ligne

Variables

TM : $Typedemessage \in [DMod, ModFin, ModRefu, ModAcc, ModSave, Error, Update]$
$IdLigne$: Id de la Ligne concernée
$User$: Id du Client

Algorithme Serveur

Require: Les operations de modifications de structure doivent être atomiques.

Reception $< TM, IdLigne, contenu >$ de $User$
if $TM = DMod$ **then**
    if $IdLigne \in Fichier$ AND $IdLigne \notin Locks$ **then**
        Ajouter $< IdLigne, User >$ à $Locks$
        Envoyer $< ModAcc, IdLigne >$ à $User$
    else
        Envoyer $< ModRefu, IdLigne >$ à $User$
    end if
end if
if $TM = ModFin$ **then**
    if $IdLigne \in Fichier$ AND $< IdLigne, User > \in Locks$ **then**
        Modifier la ligne $< IdLigne >$ avec *contenu*
        Supprimer $< IdLigne, User >$ de $Locks$
        Envoyer $< ModSave, IdLigne >$ à $User$
        Envoyer $< Update, IdLigne, contenue >$ à tous les $Clients$
    else
        Envoyer $< Error, IdLigne >$ à $User$
    end if
end if

Algorithme Client

Envoie $< DMod, IdLigne, NULL >$ au $Serveur$
Reception $< TM, IdLigne >$ du $Serveur$
if $TM = ModAcc$ **then**
    Le client modifie la Ligne, la ligne est enregistrer dans un buffer
    Envoie $< ModFin, IdLigne, contenu >$ au $Serveur$
    Reception $< TM, IdLigne >$ du $Serveur$
    if $TM = ModSave$ **then**
        Sauvegarde de la Ligne en buffer
    else
        Erreur lors de l'enregistrement
    end if
end if

Algorithm 2 Création de Ligne

Variables

TM : $Typedemessage \in [Crea, CreaR]$
$IdLignePrec$: Id de la Ligne ancre
$IdLigneNew$: Id de la nouvelle Ligne créée
$User$: Id du Client

Algorithme Serveur

Require: Les operations de modifications de structure doivent être atomiques.
Reception $< TM, IdLignePrec, contenu >$ de $User$
if $TM = Crea$ **then**
    Creation d'une Ligne après $IdLignePrec$ avec le contenu envoyé et $IdLigneNew$ généré
    Envoyer $< CreaR, IdLignePrec, IdLigneNew, contenue >$ à tous les $Clients$
end if

Algorithme Client

Envoie $< Crea, IdLignePrec, Contenu >$ au $Serveur$
Reception $< TM, IdLignePrec, IdLigneNew, contenue >$
if $TM = CreaR$ **then**
    Enregistrer localement La ligne
end if

Algorithm 3 Suppression de Ligne

Variables

Require: Les operations de modifications de structure doivent être atomiques.
TM : $Typedemessage \in [Supp, SuppR]$
$IdLigne$: Id de la Ligne
$User$: Id du Client

Algorithme Serveur

Require: Les operations de modifications de structure doivent être atomiques.
Reception $< TM, IdLigne >$ de $User$
if $TM = Supp$ **then**
    if $IdLigne \notin Locks$ **then**
        Supprimer la Ligne
        $< SuppR, IdLigneNew >$ à tous les $Clients$
    end if
end if

Algorithme Client

Envoie $< Supp, IdLigne >$ au $Serveur$
Reception $< TM, IdLigne >$ du $Serveur$
if $TM = SuppR$ **then**
    Suppression de la Ligne
end if

Chapitre 5

Voies d'amélioration

5.1 Améliorations possibles

Nous avons pensé à quelques voies d'amélioration de notre projet :

- L'ajout d'une interface graphique serait une plus-value. Dans le cadre de notre implémentation, nous pourrions possiblement utiliser un langage web ou JavaFX.
- Nous aurrions aussi pu intégrer plus de requêtes et peut-être considérer la modification d'un ensemble de ligne entier.
- Rendre l'utilsation entre client plus intéractive est aussi une possibilité : à savoir, être informé des utilisateurs qui sont connéctés ainsi que les lignes avec lesquelles ils intéragissent.
- Avoir des logs de toutes les modifications de fichier accéssible à tous pourrait être une bonne idée ainsi que le téléchargement en local du fichier.