Université de Montpellier

Projet Programmation Repartie

Naimi-Trehel

ASMA Redouane,
LEHOUAOUI Sara,
ZEDDAM Lylia


Lien GitHub

Table des matières

Chapitre 1

Introduction

Plusieurs applications réparties bénéficient des architectures en Grille, qui permettent de partager une grande variete de ressources distribuees geographiquement. Cependant, de telles applications necessitent que leurs processus puissent avoir un acces exclusif à ressources partagees. Par conse quent, l'exclusion mutuelle est un problème cruciale dans un tel contexte.
En d'autres termes, L'exclusion mutuelle assure à un processus qu'il est le seul processus du système à avoir accés à la ressource pendant un certain temps. Nous appelons le temps où le processus utilise la ressource la section critique. L'exclusion mutuelle revient donc à rendre séquentielle des acceés concurrents à une ressource partagée.
Il existe plusieurs algorithmes qui permettent de résoudre en réparti le problème de l'exclusion mutuelle. Ils peuvent être divisés en deux groupes : ceux à base de permission, et ceux à base de jeton. Le premier groupe d'algorithmes est basé sur le principe qu'un nœud entre en section critique seulement s'il a reçu la permission de tous les autres processus (ou d'une majorité d'entre eux). Dans le deuxième groupe, un unique jeton est partage par tous les sites et sa possession par un site donne le droit d'accès à la section critique. Les algorithmes à jeton sont géneralement peu coûteux en messages : au mieux la complexité en nombre de messages est O(log(n)) ou n est le nombre de sites du système.
Dans le cardre de ce projet nous proposons une impléméntation de l'algorithme de Naimi-Trehel, un algorithme à base de jeton.

Chapitre 2

Algorithme

L'algorithme Naimi-Tréhel est une brique de base pour les algorithmes distribués d'exclusion mutuelle. Basé sur la circulation d'un jeton à travers des processus, il assure l'unicité d'accès à la section critique et présente l'avantage d'avoir un faible surcoût en terme de transmission de messages.

2.1 Les bases de l'algorithme

Le point marquant de l'algorithme est dans les deux structures logiques sur lesquelles il repose. Nous les décrivons ci-après :

- La première structure est arborescente à caractère dynamique, son évolution est rattachée aux requêtes émises. La racine représente le dernier processus qui a émis une demande d'accès à la section critique. Nous appelons cette structure l'arbre Parent. Initialement, tous les processus pointent sur la même racine qui est en possession du jeton.
  Cette situation reflète le point de départ de l'algorithme. Pendant leur émission, les requêtes sont propagées le long de l'arbre jusqu'à la racine.
- La deuxième structure est une queue distribuée sur laquelle le jeton circule d'un processus à un autre. Elle se présente sous forme d'une file d'attente composée de requêtes.
  Les requêtes sont servies selon l'ordre FIFO. Nous appelons cette queue, la file Suivant.
  Soit ..pj , pj+1, .. un ensemble ordonné de processus dans cette file.
  pj pointe sur pj+1, son prochain processus dans cette file. Ainsi, pj+1 est le Suivant à avoir demandé l'accès en section critique, et c'est à lui que revient le jeton après que pj l'ait quitté.

Lorsque pj désire entrer en section critique, il envoie une demande à son Parent, pi par exemple (Comme nous l'avons indiqué, ce processus est initialement la racine de l'arbre Parent). En attendant de recevoir le jeton, pj devient racine de l'arbre parental. À partir de là, deux cas se présentent :

1. Le Parent de pj , (pi) n'est pas racine. Dans ce cas, il transmet la requête

de pj à son propre Parent. Ainsi, la requête se propage de Parent à Parent à travers l'arbre jusqu'à atteindre l'ancienne racine. Toute l'ascendance pointe sur pj qui devient le nouveau Parent.

2. Le Parent de pj (pi) est racine de l'arbre. Si pi a déjà quitté la section critique, il transmet directement le jeton à pj.
   Par contre, si pi est en section critique ou est en attente de recevoir le jeton, il pointe son Suivant sur pj dans la file Suivant. Notons qu'un processus a deux pointeurs, l'un sur son Parent dans la structure arborescente et l'autre sur son Suivant dans la file.

Chaque processus maintient des variables locales qu'il met à jour au fur et à mesure que l'algorithme évolue. Nous les définissons comme suit :

- **Info** : contient l'identifiant du site.
- **Avoir_jeton** : Variable booléenne dont la valeur est true si le processus détient le jeton, false autrement.
- **Demandeur** : Variable booléenne dont la valeur est true si le processus réclame la section critique.
- **Suivant** : Le prochain processus à entrer en section critique. La variable Suivant est nulle au début de l'algorithme, et lorsque le processus est dernier dans la file.
- **Parent** : Chaque processus a un Parent, il peut être null s'il s'agit de la racine.

Dans l'algorithme Naimi-Tréhel, les processus envoient deux types de messages :

- **Demande(j)** : Ce message est envoyé par un processus donné à son Parent, pour demander l'accès en section critique.
- **Jeton** : Ce message est envoyé par un processus à son Suivant dans la file, pour lui passer le jeton

L'exemple d'exécution illustré dans la figure 1 résume le déroulement de l'algorithme, suite à l'émission de quelques requêtes par quelques processus. Les traits en pointillés désignent la file Suivant.
Initialement, p1 détient le jeton (figure 1(a)). P1 est donc racine de l'arbre et est Parent de tous les processus restants.
p2 demande le jeton à son Parent (figure 1(b). En conséquence, p1 pointe sur le nouveau Parent p2, et positionne son Suivant sur le même processus.
À son tour, p3 demande le jeton (figure 1(c)) à p1. Par la suite, p1 transmet la requête à son nouveau Parent p2, qui à son tour met à jour ses deux variables, Parent et Suivant.
Dans la figure 1(d), p1 quitte la section critique et transmet le jeton à p2. P4 demande à son tour le jeton (figure 1(e)). Ensuite, p1 et p3 pointent leur Parent sur p4, et p3 pointe son Suivant sur p4. Finalement, p3 obtient le jeton par p2 (figure 1(f)).

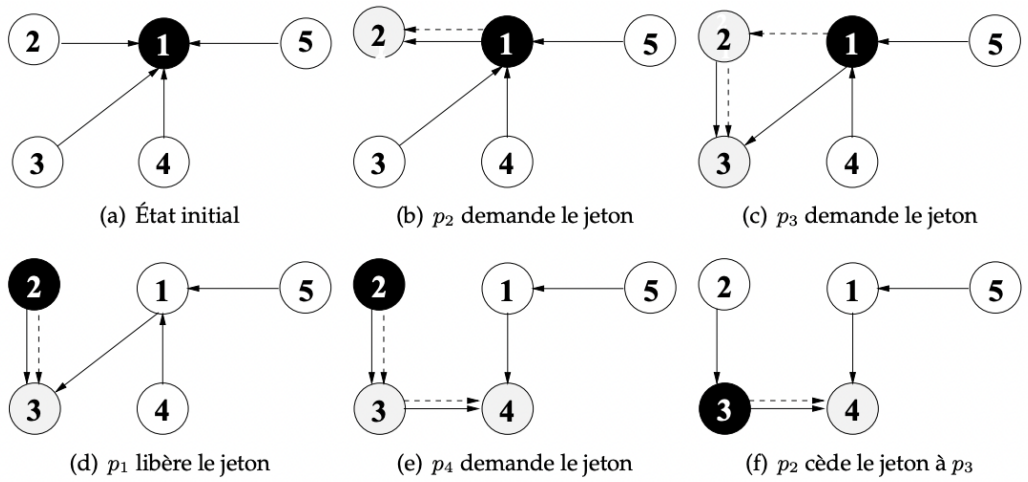

Figure 1 - Exemple d'exécution de l'algorithme Naimi-Trehel

2.2 Pseudo-code

Algorithm 1 Algorithme Naimi-Tréhel

Fonctions fondamentales de l'algorithme Naimi-Tréhel

Initialisation :
si $pj \neq racin$

$Demandeur \leftarrow racine$

$Avoir\_jeton \leftarrow false$
si non

$Parent \leftarrow null$

$Avoir\_jeton \leftarrow tru$
fin si

$Suivant \leftarrow null$

$Demandeur \leftarrow false$

Envoi d'une requête par pj :

$Demandeur \leftarrow true$
si $Parent \neq null$
pj envoie le message (request[j]) à son Parent

$Parent \leftarrow null$ pj attend le jeton
fin si

Réception d'une requête, supposons par pi le Parent de pj :
si $Parent = null$
si $Demandeur = true$ **alors** Suivant $\leftarrow$ pj
si non pi envoie le message (token) à pj
fin si
si non
pi envoie le message (request[j]) à son Parent
fin si
Parent $\leftarrow pj$

Libération de la section critique par pj :

$Demandeur \leftarrow false$ **si** $Suivent \neq null$ **alors**
pj envoie le message (token) à son Suivant
Suivant $\leftarrow null$

Chapitre 3

Organisation

3.1 Outils utilisés

3.1.1 Codeshare

Un éditeur de code en ligne

3.1.2 Overleaf

Un site utilisant le langage latex, permettant à plusieurs personne ayant l'accès à ce fichier, d'écrire en m^eme temps sur ce même dit fichier.

3.1.3 Github

Un site implémentant les fonctionnalités du logiciel Git.
Ce logiciel permet de sauvegarder les modifications apportées au code afin que les autres d´eveloppeurs du projet puissent les récupérer et à leur tour modifier le projet.

3.1.4 Discord

Discord est une plate-forme permettant aux personnes de partager et de communiquer.
Nous l'avons utilisé pour faire des réunion afin de discuter de l'avancemlent du projet.

3.2 Diagramme de Gantt

	Semaine 1	Semaine 2	Semaine 3	Semaine 4	Semaine 5	Semaine 6
Choix de l'algorithme						
Documentation						
Implementation des fonctions principales						
Implémentation du protocole TCP/IP						
Compilation + Debug						
Rédaction Rapport						

Chapitre 4

Implementation

4.1 Structure

La structure ci-dessous vient de la librairie suivante : include<netinet/in.h>, elle permet la définition des sockets. Permettant par la suite l'envoie et la réception de message via UDP/TCP.

```
struct sockaddr_in {
short sin_family;
unsigned short sin_port;
struct in_addr sin_addr;
cahr sin_zero [8]
};
```

Cette structure, décrit le type du message à envoyer ainsi que son contenu.

```
struct message {
int msg_type;
struct sockaddr_in sender;
};
```

4.2 Initialisation

Nous avons suivi l'initialisation donné dans les exercices de td :

Initialisation : choisir un sommet S quelconque et

AvoirJeton $_S$:= vrai ; Pour tout X $^?$=S, AvoirJeton $_X$:= faux ;
Pour tout sommet Y , pere $_Y$:= S ; suivant $_Y$:= NIL; Demandeur $_Y$:= faux;

Un site devra connaitre son adresse mais pas le site racine, l'adreese sera renvoyer lors de la première execution et on devra la re-saisir pour les autres sites. Pour cela on a eu besoin de trois structures :
pere, suiv et info qui représentent respectivement le père du site, le suivant et le site lui même.
On aura également besoin d'un boolean *demandeur* pour savoir si le site effectue une demande, *avoirjeton* pour savoir si le site possède le jeton, *is-set-pere et is-set-suiv* pour savoir si le site a un père (et donc si c'est la racine) et un suivant.
Mais aussi d'une variable mutex et une variable conditionnelle pour les threads.
Pour distinguer la racine des autres sites, le nombre d'argument diffère, la racine n'aura qu'un numéro de port et un entier 0. La racine est initialisée sans pere, sans suivant, possède le jeton et n'est pas demandeur.
Les autres sites ont un père (*is-set-pere* est vraie), n'ont pas de suivant et ne possèdent pas le jeton.

4.3 Communication entre sites

Pour la communication entre sites, nous avons choisi le protocole TCP/ IP. En effet les processus communiquent entre eux et doivent faire des envois de message ainsi que le jeton et pouvoir les receptionner. Chaque site est donc client et serveur à la fois. De plus ce protocole a un très haut niveau de fiabilité lors du transfert de données. Même si un paquet d'informations est perdu en suivant un canal particulier, le protocole garanti qu'il arrivera à destination en empruntant un autre chemin.
Un autre avantage significatif du TCP/IP réside dans le fait que la plupart des protocoles qui le composent sont sans état, ce qui signifie que les informations relatives à la session de chaque intervenant d'une communication n'a pas besoin d'être retenue par le serveur, ce qui permet à tous d'utiliser en permanence les chemins libres d'un réseau.

Chapitre 5

Conclusion

On a un programme fonctionnel, où plusieurs sites peuvent être lancés, faire une demande d'accés à la section critique et sont gérés d'après le principe de Naimi-Trehel en utilisant principalement que deux structures.