HMIN105M
Principe de la programmation concurrente

Projet

Master 1 - AIGLE
Faculté des sciences de Montpellier
Année universitaire 2020/2021

Table des matières

1. Conception **2**
Composition du projet 2
Structure de l'état du système 2
Utilisation des sémaphores 4
Choix des limites 4
. 4

Utilisation **5**
Implémenté : 5
Non implémenté : 5
Avertissement : 5
Compilation & exécution : 5
Nota Bene 5

1. Conception

Composition du projet

Pour ce projet, nous avons un programme centralisé en trois fichiers. Un serveur, un client et un fichier texte contenant les différentes ressources à disposition.

Le fichier des ressources est écrit de la façon suivante :

- une ville par ligne
- par ligne (séparé par des espaces) : <nom_ville> <nombre_cpu> <nombre_stockage>

```
exemple du contenu de fichier.txt :

Montpellier 56 1000
Lyon 69 1500
...
```

Le serveur, lui, se charge de lire ce fichier, puis créer la zone de mémoires partagées, dont la taille dépendra du nombre de villes (appelés stores dans le code). Il initialisera aussi le tableau de sémaphores et à la toute fait, lorsque l'utilisateur le voudra, il se détachera de la zone de mémoire partagée, la supprimera et de même pour le tableau de sémaphores.

Enfin, le client. Il est composé de trois threads. Un pour l'attente en continu des mises à jours des ressources disponibles. Un second qui effectuera les traitements sur les sémaphores et la zone de mémoires partagées en fonction de la demande client. Et enfin un troisième qui s'occupe des demandes du client.

Structure de l'état du système

Notre zone de mémoire partagée sera utilisée afin de stocker les données systèmes ainsi que les évènements *(logs)* qui ont eu lieu. Pour gérer au mieux nous avons décidé de la décomposer en différentes structures que nous allons présenter ci-dessous étapes par étapes.

Comme nous l'avons dit précédemment, nous avons omis que notre état système sera composée de donnée et d'évènements. Nos données seront représentés, comme dans l'énoncé, avec la ville, nombre de CPU et un espace de stockage (en Go). Pour les évènements, représentant les réservations omises par un utilisateurs, nous auront les même données vu précédemment avec un nom, un prénom, un drapeau *(flag)* - définit si la demande était en mode exclusive ou non - et un nombre d'utilisateurs - définit, si en mode exclusif, le nombre d'utilisateurs qui partagent ses données -. Pour mieux visualiser en voici une représentation.

```c
typedef struct store {
  int city;
  int cpu;
  int go;
} store;
```

```c
typedef struct log {
  id userid;
  char name[20];
  char firstname[20];
  int city;
  int cpu;
  int go;
  int users[MAX_USER];
  int flag;
} log;
```

Finalement en regroupant ces deux structures nous obtenons une structure générale représentant les données présentent dans notre zone de mémoire partagée :

```c
typedef struct shm_data {
  int nb_client_connected;
  int nb_stores;
  store stores[MAX_STORES];
  int nb_logs;
  log logs[MAX_CLIENT * MAX_STORES];
} shm_data;
```

Dans le client, nous avons un structure pour le message, contenant elle même un tableau de structure des commandes (order) qui est similaire à store.

```c
typedef struct{
  user user;
  int size;
  order* orders;
  int flag; // pour mode exclusif, pour mode partagé
} message;
```

Utilisation des sémaphores

Nous utiliserons un seul tableau de sémaphores de taille $(4 * nb\_stores)$. Pour chaque ville, nous accordons 4 sémaphores chacune, la première représente le nombre de CPU disponibles, la seconde représente le nombre de CPU en mode partagé utilisée, la troisième représente le nombre de Go disponibles et pour finir la dernière sera le nombre de Go en mode partagé utilisée.

Enfin, un dernier pour l'actualisation des ressources disponibles chez les clients. Un thread "on_update" tournera en boucle, attendant que ce sémaphore soit au moins à 1 (opération P décrémentant de 1) pour afficher les ressources après une modification (affectation des ressources demandées / rendue de ressources). Juste après une modification, le client l'effectuant, fait une opération V incrémentant ce sémaphore du nombre de clients connectés à la zone de mémoire partagée, permettant à tous d'avoir les nouvelles ressources.

Un schéma vaut mieux qu'un long discours, voici un exemple :

```
// 1er magasin (store)
struct storeA;
store.city = Montpellier;
store.cpu = 56;
store.go = 1000;

// 2eme magasin (store)
struct storeB;
store.city = Lyon;
store.cpu = 69;
store.go = 1500;
```

```
// Représentation du tableau de sémaphores à
    l'initialisation
sem[0] = 56
sem[1] = 0
sem[2] = 1000
sem[3] = 0
sem[4] = 69
sem[5] = 0
sem[6] = 1500
sem[7] = 0
sem[8] = 0 // Semaphore utile pour
    l'actualisation des ressources chez tous
    les clients
```

Choix des limites

Pour créer notre zone de mémoires partagées, nous avons dû fixer certaines limites afin d'éviter tous problèmes de dépassements de mémoires ou avoir des pointeurs à l'intérieur de la zone ou autres. Ces limites sont représentées par des variables globales, elles peuvent donc être modifiées.

Nous avons donc, une limite de dix stores maximum, de même pour le nombre de clients pouvant se connecter en même temps.

Utilisation

Implémenté :

- Connexion d'utilisateur.
- Commande/Libération de données.
- Possibilités de faire plusieurs demande en une seule commandes.
- Réception de toutes les données lors d'une nouvelle mise à jour.

Non implémenté :

- mode partagée (semi-implémentée).
- TCP.

Avertissement :

- !! Ne surtout pas donner de valeur < 0 pour les CPU ou les Go!!
- Pendant la phase de "connexion" vous ne verrais pas les mises à jour s'effectuer.
- Si pendant votre saisie, un affichage survient en plein milieu, faite comme si de rien n'était.
- Que l'on dise oui ou non, afin de recommander, celui-ci demande en boucle, pour quitter un client utilisé CRTL + C.

Compilation & exécution :

```
#création des dossiers bin et obj
#A n'exécuter qu'une seule fois!
make init

#Si les dossiers ne se sont pas créés
sudo mkdir bin obj

#Compilation
make

#Lancement du serveur
./bin/server datacenter

#Lancement d'un/des clients
./bin/client
```

Nota Bene

Nous avions commencé à faire la partie TCP, celle-ci étant en grande partie terminée, nous n'avons pas pus la finaliser car nous avons eu du mal à implémenter la version centralisée, sur laquelle nous nous sommes donc réorganisé.