HMIN105M - projet
Système de réservation distribué

A déposer au plus tard le vendredi 4 décembre 2020 à 18h

Instructions : Ce projet est à réaliser en binôme. Lire attentivement l'énoncé avant de commencer le travail. Pour la mise en place des connexions et communications distantes, vous utiliserez le protocole TCP/IP et les fonctions C utilisées en cours. Le plagiat est strictement interdit.

L'idée de ce projet s'inspire d'un système de réservation de ressources de calcul et/ou de stockage sur une plateforme de grille ou de cloud. Il a pour objectif de permettre à des clients de louer des puissances de calcul ou des espaces de stockage distants répondant à des besoins spécifiques (exemple : exécuter une simulation scientifique sur une architecture distribuée de processeurs et de mémoire pour stocker (le temps de la location) les données traitées et produites). Dans le système à mettre en oeuvre, un client aura la possibilité de louer des ressources, soit en mode exclusif (les ressources louées sont utilisées par un seul client pendant toute la durée de la réservation), soit en mode partagé (les ressources louées peuvent être utilisées en même temps par plusieurs clients).

L'application à réaliser doit répondre à des contraintes définies. Elle est constituée d'un serveur concurrent et de plusieurs clients. Le rôle du serveur est de mettre en place un espace partagé représentant l'ensemble des ressources disponibles et louées (en d'autres termes, l'état des ressources du système), de gérer les accès clients à cet espace (pour louer/libérer des ressources), de maintenir cet espace dans un état cohérent et de diffuser toute modification à tous les clients. Le rôle des clients est d'échanger avec le serveur pour effectuer des réservations et libérations de ressources et pour avoir en continu une vue de l'état global de toutes les ressources offertes par le système de réservation.

Plus précisément :

1. Le serveur concurrent est constitué d'un processus parent qui est responsable de la mise en place de l'état initial du système de réservation (l'état est défini plus loin) et d'attendre les connections des clients. Le traitement de chaque client est délégué à un processus fils qui lui sera exclusivement dédié. Ce processus fils, commence par envoyer l'état courant du système à son client, ensuite et en boucle, attend toute demande de réservation ou de libération de ressources par ce client, traite cette demande, met à jour l'état des ressources concernées et déclenche une diffusion du nouvel état à tous les clients. En parallèle, le processus fils doit pouvoir envoyer à son client, toutes les modifications issues des autres clients. Des threads sont à utiliser pour mettre en oeuvre le comportement parallèle au sein d'un processus fils pour le traitement de son propre client. Remarque : un processus fils ne peut communiquer qu'avec son propre client. Il sera nécessaire d'utiliser un ou des tableaux de sémaphores pour gérer les communications entre processus fils du serveur. Il sera aussi nécessaire de veiller à avoir un état cohérent de l'état des ressources à tous moment (attention aux accès concurrents).
2. Le client doit avoir la possibilité de visualiser l'état du système de réservation dès sa connexion au serveur. Ensuite et en boucle, le client peut envoyer des demandes de réservation ou de libération de ressources au serveur. Ces demandes sont à saisir au clavier. Si une demande de réservation de ressources ne peut être entièrement satisfaite, le client doit bloquer jusqu'à satisfaction de sa demande (autrement dit : le client doit obtenir

tout ce qu'il demande en même temps, sinon il est mis en attente). En parallèle, le client doit être capable de recevoir automatiquement et en continu (sans avoir à le demander) l'état des ressources après chaque mise à jour faite sur le serveur et de l'afficher. Ce comportement parallèle sera mis en oeuvre en utilisant les threads.

3. L'état du système de réservation représente l'état des ressources. Le système est constitué de plusieurs sites géographiquement distribués (exemple Lyon, Montpellier, Rennes, Toulouse), chaque site propose un nombre de processeurs et un volume de stockage (exemple : {{Lyon : 64 CPU et 2000 Go}, {Montpellier : 92 CPU et 700 Go}, ...}. L'état du système doit décrire pour chaque site et chaque type de ressource (CPU ou volume de stockage) la quantité réservée par utilisateur et le mode de réservation (en exclusif ou en partage) (exemple : {Lyon : 60 CPU et 1000 Go libres, réservations exclusives : {3 CPU et 1000 GO par l'utilisateur Toto, 1 CPU par l'utilisateur Titi}, réservations partagées : {aucune } }). Un client doit pouvoir demander une réservation de ressources impliquant plusieurs sites (exemple : un utilisateur Toto demande 2 CPU à Rennes plus 10 CPU et 100 Go de stockage à Toulouse, le tout en mode exclusif). Si un client est bloqué en attente de satisfaction de sa demande, un autre client doit pouvoir réserver des ressources : si sa demande peut être satisfaite, elle ne doit pas être mise en attente. L'état du système et sa gestion sont à mettre en oeuvre via un segment de mémoire partagée et un ou des tableaux de sémaphores. Vous êtes libres de choisir la structure de données représentant l'état du système à partir du moment où toutes les informations citées plus haut sont prises en compte. L'utilisation des files de messages n'est pas utile et donc ne sont pas à utiliser.

Le projet est organisé en plusieurs étapes. Chaque étape mérite d'être bien réfléchie avant d'être mise en oeuvre, sans oublier de faire le lien avec les autres étapes.

1. Définir l'architecture de votre application (structure de l'état du système, objets IPC utilisés et principe de leur utilisation, les processus et threads composant l'application et le rôle de chacun, le protocole d'échange à distance entre un client et le serveur). Remarque : après chaque mise à jour de l'état du système, il est possible d'envoyer uniquement la modification et non l'état en entier des ressources (à vous de faire le bon choix). Cette étape est à résumer dans un document pdf qui sera à déposer avec le code source.
2. Réaliser une version centralisée / sans communications distantes : dans cette partie, il s'agit de mettre en place l'architecture concurrente du serveur sans lien de parenté entre processus le composant. Le comportement d'un client sera intégré dans un processus remplaçant un processus fils. Plus précisément :
   — Un processus serveur (inspiré du processus parent du serveur) mettant en place le système de réservation
   — des processus clients (inspirés des processus fils du serveur et du processus client) où chaque processus doit afficher le contenu de l'état des ressources, ensuite et en boucle, attendre une saisie de l'utilisateur, traiter la demande utilisateur, mettre à jour l'état des ressources, notifier cette modification à tous les processus pour qu'ils affichent le nouveau contenu à leur utilisateur. Bien entendu, en parallèle, un processus doit pouvoir afficher à son utilisateur, toutes les modifications effectuées par les autres utilisateurs/processus.
3. Une fois la version centralisée terminée et testée, garder une copie. Modifier maintenant le code pour mettre en place le serveur concurrent décrit dans l'énoncé (parent et fils) et des clients distants :
   — le serveur parent créera un fils à chaque connexion d'un nouveau client.
   — la saisie au clavier d'une demande utilisateur se fait désormais par un programme client qui communique à distance avec un processus fils du serveur (transformer la saisie dans le processus de la version centralisée par une réception de message et créer le lient de parenté avec le parent).

— l'affichage de l'état du système se fait dans le programme client. Il est judicieux d'avoir aussi un affichage coté serveur pour vérifications.
— etc.

Quelques remarques :

— la diffusion d'une mise à jour de l'état du système implique des envois parallèles de cette mise à jour aux différents clients.
— la gestion des erreurs est indispensable. Exemples : si un objet IPC est supprimé les processus doivent quitter proprement ; si un processus fils est tué, l'exécution doit pouvoir se poursuivre sans erreurs (traitement des autres clients) ; en cas de déconnexion ou fermeture d'une socket, un processus concerné doit quitter proprement.
— Bien prendre en compte l'ergonomie de votre application (elle sera installée et exécutée par des utilisateurs qui n'auront pas à lire le code avant de comprendre les fonctionnalités réalisées)
— Il n'est pas demandé de réaliser une interface graphique.

Dépôt de votre travail

1. Le code source à remettre sera le code le plus fonctionnel (soit l'application centralisée, soit l'application distribuée). Le code source doit être positionné dans un répertoire "projet", incluant aussi un Makefile.
2. Créer une archive projet.tgz contenant votre répertoire "projet", le fichier pdf (première étape) et un fichier README décrivant le travail effectué et donnant les instructions nécessaires pour la compilation et l'exécution.
3. Désigner un membre de votre binôme pour déposer l'archive sur Moodle : HMIN105M : atelier projet (donc un seul dépôt par binôme). Le dépôt doit être anonyme : le contenu de vos fichiers, les noms des fichiers, etc. ne doivent contenir aucune information permettant d'identifier les propriétaires du travail remis.
4. Le dépôt est possible avant vendredi 04/12 à 18h pour procéder ensuite à une évaluation par les pairs .