Travail pratique #2 - IFT-2245

Jérémi Grenier-Berthiaumet et Olivier Lepage-Applin

19 mars 2019

Côté client : Sémaphore pour tenter d'éviter le busy waiting sur la variable 'count'. Fonction 'st_signal()' enlevée car le démonstrateur a dit qu'elle ne servait à rien. Il a fallu rajouter des 'free()' dans les 'main' pour libérer proprement leur 'malloc'. Plusieurs mutex au lieu d'un seul pour que le traitement soit plus efficace (meilleur parallélisme). Tous les compteurs sont protégés par leur propre mutex, mais on n'avait pas de besoin d'un mutex pour ce qui est touché par un unique thread (i.e. le count++ dans ct_init) Certaines macros pour la lisibilité et rapidité de changement durant développement. Nouvelle méthode 'ct_init_server()' qui permet d'initialiser le serveur de manière plus efficace. Notre solution précédente était d'utiliser un mutex ainsi qu'un booléen dans 'ct_code()', mais on a alors autant de threads que de clients qui attendent que ce qu'un seul thread peut faire soit terminé. Désormais, on utilise un booléen pour tout annuler en cas d'erreur durant l'initialisation du serveur. Détection de la dernière commande à l'intérieur du 'for' parce qu'on dirait bien qu'on n'était pas supposé écrire à l'extérieur des TODO. Évitement des malloc le plus possible (en passant en paramètre des pointeurs de variables déclarées localement). La variable 'nb_registered_clients' est utilisée pour attendre que tous les clients fassent leur 'CLO' lors d'un appel à 'END'. Les variables de comptage en lien avec 'REQ' sont légèrement ouvertes à l'interprétation. Pour notre part, s'il y avait un nombre négatif qui indiquait le nombre d'arguments d'une requête 'REQ' celle-ci n'est pas réellement comptée. Cependant, si c'est un nombre d'argument positif mais qui ne correspond pas à "num_resources + 1", alors celle-ci est comptée comme une erreur. Signature de la fonction 'send_request()' a été modifiée pour avoir accès au struct 'client_thread'. Pas nécessaire, mais plus "clean".

Côté serveur : On a conservé la fonction qui ne reçoit que le header en premier, et appelons ensuite une fonction qui traite les arguments ensuite : cela nous permet de connaître la grosseur du buffer à remplir par la fonction qui lit le contenu du socket. Utilisation d'un array de fonction pour gérer de façon élégante les différents comportements du serveur.

Protocole : On utilise 'id' plutôt que 'tid' pour les communications. On a assumé que les 'WAIT' ne sont des réponses qu'aux 'REQ' des clients.

Général : Les démonstrateurs nous ont dit qu'il était correcte de simplement 'exit' et de ne pas tenter de 'free' lors d'un erreur sur 'malloc'. Un démonstrateur a aussi mentionné qu'il ne serait pas nécessaire de faire la vérification sur la variable de retour du 'close(socket)'. On nous a dit de préciser cela : nous ne considérons pas que la requête 'CLO' fait partie du nombre de requêtes que devrait envoyer un client. De plus, une requête 'REQ' qui est réenvoyée à cause d'un 'WAIT' n'est pas comptée comme étant une requête de plus (pour la statistique de 'count_request') (todo : verify/clarify this claim).

Algo du banquier : (oli)