Documentation technique

Ce programme a pour de mesurer les performances de l'ordinateur sur lequel il est lancé. Par la suite les statistiques de l'ordinateur sont envoyées à une base de données qui va permettre de comparer les différents scores des utilisateurs puis d'établir un classement.

Benchmark en langage C

## Fonctions relatives à la base de données

**void createDatabase();**

La fonction createDatabase est une fonction standalone qui sert à recréer une base de données.

**void createUser(char \*username);**

La fonction createUser permet la création d’un utilisateur dans la table user, elle prend en argument une chaine de caractère de maximum 60 caractères.

On fait d’abord un select de la base pour vérifier si le username renseigné n’existe pas déjà en base, s'il existe déjà on ne créé pas le user et on informe l’utilisateur que le username renseigné existe déjà. Sinon si l’user n’existe pas on le créé en base.

**void getUser(char \*username);**

La fonction getUser permet de consulter la table user et d’afficher les informations d’un utilisateur, elle prend en argument une chaine de caractère de maximum 60 caractères.

On fait un select de la base pour récupérer les informations du user demandées et les afficher. Si l' user n’existe pas on informe l’utilisateur que le username renseigné n’existe pas.

**int getId(char \*username);**

La fonction getId permet de récupérer en base l’ID du user renseigné en paramètre pour les fonctiosn suivantes qui ont besoin de cette information pour leur requête, si le user n’existe pas on informe l’utilisateur que le user demandé n’existe pas sinon on récupère son ID.

**void createOrUpdateResult(int choiceLeaderboard, char \*username, int testValue);**

Cette function permet de déterminer si pour les résultats venant des tests de performance on doit créer une nouvelle ligne en base car c’est la première fois que l’utilisateur fait un test ou si il faut mettre à jour ses résultats. On fait donc une première requête avec son ID recupéré avec getId() pour vérifier si son ID existe dans la table result, si on a un retour c’est que l’utilisateur a déjà des résultats en base, on lance donc la fonction updateResult() sinon on lance createResult.

Le premier argument choiceLeaderboard, indique quel résultat on veut ajouter/modifier (1-cpu, 2-ram, 3-écriture, 4-réseau, 5-global), le second argument est le username concerné par l’ajout/modification et le troisième argument testValue est la valeur retournée par le test de performance.

**void updateResult(int choiceLeaderboard, char \*username, int testValue);**

Cette fonction permet donc l’update de la table result si le user rentré en argument a déjà des résultats en base

**void createResult(int choiceLeaderboard, char \*username, int testValue);**

Cette fonction créée dans la table result une ligne pour le nouveau user qui n’avait encore aucun résultat en base.

**void getLeaderboard(int choiceLeaderboard);**

Cette function permet un classement des 10 user avec le score le plus élevé dans un test, l’argument choiceLeaderboard permet de choisir quel leadearboard on veut afficher

## Fonctions relatives à la partie réseau

**Choix du protocole :**

Le protocole UDP, qui est un protocole sans connexion, nous permet d’envoyer et de recevoir des paquets rapidement. Même si le protocole UDP est moins fiable que le protocole TCP vis-à-vis de la perte de paquets, il permet d’envoyer et de recevoir des paquets plus rapidement car il ne nécessite aucune connexion. Dans l’optique de tester la vitesse de connexion d’un ordinateur, ce protocole nous permet un traitement plus rapide.

Le serveur utilisé pour le programme, tourne indéfiniment et renvoi au client les paquets qu’il a reçu d’eux.

**void startWSAS ();**

Cette fonction sert à initialiser l’utilisation de la librairie winsock2.h

**int sendToServer(SOCKET sockclt, SOCKADDR\_IN \*sin, char \*buffer) ;**

Cette fonction permet d’envoyer le contenu du buffer au serveur, elle prend en paramètre le socket de la machine cliente, la structure (SOCKADDR\_IN) qui contient l’IP et le port du serveur sur lequel envoyer les données. En dernier paramètre est indiqué le buffer qui contient les données à envoyer.

**float fTimer(char action, float starter);**

Cette fonction permet de calculer le temps écouler entre deux instructions.Elle prend en paramètre une variable qui contient le résultat. Le deuxième appel de la fonction permet de retourner le résultat de la différence entre le premier appel de la fonction et le dernier appel.

**int calculatePing();**

Cette fonction envoi un paquet UDP vide au serveur qui va à son tour renvoyer un paquet vide au serveur. Cette fonction a pour but de déterminer le temps que met le client a envoyé et récupérer un paquet en millisecondes.

**float sendPacketToServer(int packetsize);**

La fonction sendPacketToServer prend en paramètre la taille du paquet à envoyer au serveur.

Dans le programme, cette fonction envoie 4000 paquets de 10Ko au serveur qui renvoi à chaque fois au client le paquet reçu.

La fonction retourne le temps prit en millisecondes pour envoyer et récupérer le paquet. Le résultat de la fonction est ensuite converti en Mb/s.