DOCUMENT D'ARCHITECTURE

LE SERVEUR

Le serveur est composé de 4 documents : header.h main.c, fonction.c et parsing.c

Le header.h en plus de contenir tous les prototypes des fonctions utilisées dans tout le code, contient deux structures :

Une structure « parametre » :

Lors du lancement du serveur vous avez la possibilité de renseigner plusieurs paramètres. Ceux-ci seront stockés dans cette structure afin d'être réutilisé plus simplement dans le reste du code.

Une structure User :

Le serveur doit pouvoir accueillir et gérer plusieurs clients.

Pour ceci, il utilise une liste chainée. Chaque User à son propre socket de discussions avec le serveur ainsi qu'un compteur de pixels et un chronomètre afin de pouvoir limiter le nombre de pixels posés par le temps imposé lors de la création du serveur.

LE FICHIER MAIN :

Ce fichier est le cœur du projet.

Lors du lancement du serveur, les paramètres rentrés sont récupérés et enregistrés dans une structure paramètre grâce à la fonction paramétrage.

Si aucun paramètre n'est renseigné, la structure est initialisée avec les valeurs suivantes :

Port = 5000

Limite de pixel par minute = 10

Dimensions = 80x40

Après avoir récupéré les paramètres une matrice est créée à partir de ceux-ci. La matrice est remplie de « //// » ce qui correspond à la couleur blanche en base64.

La matrice est un tableau 3 dimensions, dont la 3ème est un tableau de char. On représenter la matrice comme ceci : [[« chaine de caractères »] ; [«chaine de caractères »] ; …]

Le serveur est créé avec un point de rencontre local et une socket d'écoute.

Une structure pollfd « poll_accept » est initialisée pour utiliser la fonction poll, ce qui permet de capter les évènements se passant sur le point de rencontre locale.

En plus de cela, un tableau de poll est créé.

Si un client se connecte sur la socket d'écoute, le « poll_accept » est déclenché et on peut appeler la fonction « ajout_client ».

Cette fonction permet d'initialiser un client dans une structure User et de l'ajouter à la liste chainée.

Pour cela, elle utilise la fonction « accept » afin d'initialiser une « salle de discussion » ou un « point de rencontre distant » sur une socket différente que celle d'écoute.

En plus de cela, la liste de poll « poll_message » est incrémenté et la socket de discussion du client est rattaché au dernier pollfd ajouter dans la liste.

NB : En temps normal, l'appel de la fonction « accept » est bloquant ce qui empêcherait toutes discussion entre les autres clients et le serveur, car celui n'exécuterait pas le reste de son code tant que personne ne souhaite se connecter au serveur.

C'est pour cela que l'utilisation de la structure pollfd est importante.

Après que le serveur est accepté un client, ou s'il n'en a pas à accueillir, il peut s'occuper de traiter les messages des clients déjà présents grâce à la fonction « reception_message »

La fonction « reception_message » utilise la fonction « read » qui a aussi un appel bloquant.

Afin d'éviter ce blocage, on va parcourir la liste de poll « poll_message » et regarder si l'un de ces éléments a été enclenché. Et c'est seulement dans ce cas qu'on appelle la fonction read qui prend en paramètre le « file descriptor » du poll enclenché, car il réfère sur la socket de discussion du client qui a envoyé le message.

En fonction de la valeur retournée par la fonction « read » on ferme le serveur parce qu'il y a une erreur, ou on supprime le client de la liste-chainé ainsi que son poll grâce aux fonctions « supprime_client » et « supprime_poll » car il s'est déconnecté du serveur.

Si l'on ne rentre pas dans ces cas, c'est que le client nous a fait une requête, donc on traite son message avec la fonction « traite_message ».

La fonction « traite_message » prend en paramètre le message envoyé par le client pour l'analyser.

On commence d'abord par découper le message en fonction des espaces (' ') qu'il contient grâce à la fonction « strtok ».

Cette fonction nous renvoie le premier « token » avant le délimiteur spécifié (espace dans notre cas), on peut donc le comparer avec les types requêtes que peut traiter notre serveur.

Ainsi le 1er token sera égal à une des string suivantes : /setPixel ; /getMatrix ; /getSize ; /getVersion ; /getLimits ; /getWaitTime.

Si ce n'est pas le cas, le serveur informe le client qu'il ne connait pas la requête qu'il demande.

Pour les requête /getSize, /getLimits et /getVersion le serveur renvoie les paramètres avec lesquels il a été initialiser.

Pour la requête /getWaitTime, on calcule la différence de temps entre l'instant présent et la dernière fois que le chrono du client a été enclenché. Puis on renvoie le résultat.

Pour la requête /getMatrix, on vient récupérer toutes les strings de la matrice du serveur passer en paramètre de la fonction « traite_message ». Toutes ces chaines de caractères sont collées les unes aux autres pour n'en former qu'une seule qui sera envoyée au client.

Si le client envoie /setPixel, on vient récupérer les paramètres qu'il a mis en utilisant « strtok ».

Enfaite on récupère les tokens restant après le délimiteur espace (' ').

On compare la position du pixel par rapport aux dimensions de la matrice. Si la position les dépasse, le serveur renvoie l'erreur 'Out of Bound'.

Dans le cas contraire on regarde si le client a le droit de déposer un pixel ou s'il dépasse la limite imposée grâce à la fonction « check_limite ». Si c'est le cas le serveur renvoie « Out of quota ».

Dans le cas contraire, le pixel est déposé sur la matrice.

Comment fait le serveur pour savoir quel client fait la requête et ainsi vérifier s'il ne dépasse pas la limite de pixel par minute ou encore lui répondre ?

Dans la fonction « reception_message », on appelle la fonction « traite_message » qui prend en paramètres le message envoyé sur la socket surveillé par le poll qui a été enclenché, le poll en question, les paramètres du serveur, la matrice et le client.

Le client passé en paramètre est récupéré dans la liste-chainée « User » grâce à la fonction « select_client ». Celle-ci va parcourir toute la liste-chainée et comparer la socket de chaque client avec la socket surveiller par le poll qui a été enclenché. Si les socket sont égales, la fonction renvoie les client dont la socket correspond à celle surveillé par le poll.