Chat Room (Socket en C)

LAHMAM Lamia
LABRIJI AMINE
BOUKHRIS Chouaib

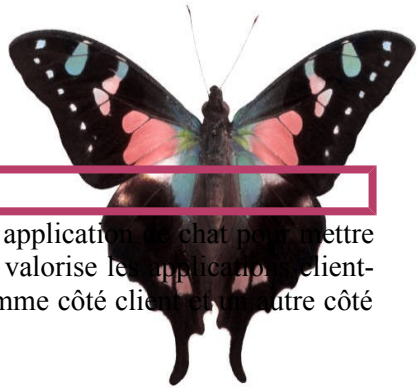
Responsable du projet:

Professeur M.ElAnsari.
melansari@yahoo,fr

REMERCIEMENTS

Nous tenons à adresser nos profonds remerciements à Monsieur Mel.ANSSARI professeur au département informatique à la Faculté des Science Ben M'Sik, qui a proposé le sujet et à bien voulu diriger ce travail avec ses très grandes compétences reconnues non seulement sur le plan professionnel mais aussi scientifique.
Nos remerciements vont également à tous ceux et celles qui m'ont aidé de près ou de loin à la réalisation du projet.

TABLE DES MATIÈRES



PRÉSENTATION

Le projet mis en œuvre est intitulé « Chatroom » invite a réaliser une petite application de chat pour mettre en œuvre les techniques de programmation avec les sockets. Ce mini projet valorise les applications client-serveur. En fait pour réaliser une application de chat il faut écrire un programme côté client et un autre côté serveur.

ChatRoom est un salon de discussion ou un clavardoir (au canada1) est un lieu de rencontre virtuel, accessible à partir d'un site, que l'internaute peut choisir, selon le sujet proposé ou l'intérêt du moment, afin de dialoguer par clavier interposé, avec un nombre relativement restreint de participants.

IDENTIFICATION DES OBJECTIFS DU PROJET

Réaliser une petite application de chat pour mettre en œuvre les techniques de programmation avec les sockets. Ce mini projet valorise les applications client-serveur. En fait pour réaliser une application de chat il faut écrire un programme côté client et un autre côté serveur. Le client peut se connecter au salon du chat en saisissant un nickname différent des nicknames des chatteurs présents dans le salon.
Le serveur, jouant le rôle d'un réflecteur, reçoit les messages envoyés par les clients et les diffuse vers tous les chatteurs en ligne.
Une fois connecté, le client est invité à saisir son message ou à quitter le salon en tapant « exit ».
Tous les messages seront affichés sur la console pendant le chat. Pour assurer la saisie du message, son envoie vers le serveur et la réception des autres messages provenant de la part du réflecteur, le client doit disposer d'au moins deux threads indépendants.
Le premier se charge d'écouter les paquets UDP venant du serveur et les afficher sur le terminal.
Le deuxième récupère et envoie vers le serveur les messages saisis par l'utilisateur.
A chaque fois que le serveur reçoit une requête, il crée un thread qui va analyser cette requête et fournit la réponse adéquate qu'il envoie aux destinataires concernés.
L'utilisation de plusieurs threads dans le programme du serveur est utile dans le cas où le serveur est assez chargé.
En effet ces threads lui permettent de traiter plusieurs requêtes en parallèle.
Nous disposons de quatre procédures permettant la réalisation des différentes fonctions gérées par le serveur (enregistrement d'un nouveau client, diffusion des messages vers l'ensemble des chatteurs en ligne, mettre fin à une connexion d'un client voulant quitter le salon du chat et fournir aux clients la liste des chatteurs connectés au chat).

FONCTIONNALITÉ TECHNIQUE

LANGAGE UTILISÉ :

C'est qualifié de langage de bas niveau dans le sens où chaque instruction du langage est conçue pour pouvoir être compilée en un nombre d'instructions machineassez prévisible en termes d'occupation mémoire et de charge de calcul. Il propose un éventail de types entiers et flottants conçus pour pouvoir correspondre directement aux types supportés par le processeur. Il fait en outre un usage intensif de la notion de pointeur. Il a une notion de type composé, mais ne propose aucune opération qui traite directement des objets de plus haut niveau (fichier informatique, chaîne de caractères, liste…). Ces types plus évolués doivent être traités en manipulant des pointeurs et des types composés. De même, le langage ne propose pas en standard la gestion de la programmation orientée objet, ni de système de gestion d'exceptions, ni la programmation parallèle. Il existe des fonctions standards pour gérer les entrées-sorties et les chaînes de caractères, mais contrairement à d'autres langages, aucun opérateur spécifique pour améliorer l'ergonomie. Ceci rend aisé le remplacement des fonctions standards par des fonctions spécifiquement conçues pour un programme donné.

Le langage C a été inventé pour écrire le système d'exploitation UNIX. Il a conservé de cela une très grande efficacité pour tout ce qui concerne le développement système. Ainsi le noyau de grands systèmes d'exploitation comme Windows et Linux sont développés en grande partie en C.

OUTILS UTILISÉ :

Code::Blocks est un environnement de développement intégré libre et multiplateforme. Il est écrit en C++grâce à la bibliothèque wxWidgets. Pour le moment, Code::Blocks est orienté C et C++, mais il supporte d'autres langages comme le D.

Code::Blocks est développé pour Linux, Windows et Mac OS X, mais les utilisateurs ont réussi à compiler le code source sous FreeBSD.

Code::Blocks se veut simple d'utilisation, mais peut se révéler très complet si on va fouiller un peu dans les options. Il est très personnalisable, et extensible, grâce à son architecture de plug-ins, dont la plupart sont inclus dans l'archive et l'installateur. Vous n'aurez donc pas à les installer un à un.

FONCTIONNALITÉ ET CODAGE

FONCTION 1:

chat_join(Client client)

DESCRIPTION :

La procédure chat_join prend en paramètres le nickname du client et son socket d'émission de ses messages. Elle consulte ensuite la liste des chanteurs en ligne qui est enregistrée dans un fichier. Deux cas sont envisageables, si le nickname existe dans cette liste le serveur envoie au client une invitation à saisir un autre nickname sinon l'application enregistre le nouveau chateur en ajoutant son nickname à la fin du fichier et en lui envoyant un message de bienvenue dans le salon de chat et la liste des chanteurs en ligne. Dans les deux cas le serveur envoie la réponse vers la socket du client .L'adresse du client est récupérée à partir de sa socket d'émission.

CODE :

(Voir Le code source si joints)

FONCTION 2:

chat(Client *clients, Client sender, int actual, const char *Buffer, int from_server)

DESCRIPTION :

Cette procédure permet la transmission du message vers tous les chateurs. Sauf l'expéditeur du message.

CODE :

(Voir Le code source si joints)

FONCTION 3 :

chat_leave(Client *clients, int to_remove, int *actual)

DESCRIPTION

Suite à la demande du client de quitter le salon de chat, cette procédure supprime son nickname de la liste des chateurs enregistrée dans le fichier. La suppression d'un nickname de notre fichier nécessite le recours à un fichier temporaire où on copie tous les nicknames sauf celui qui désire partir. A la fin de la procédure on détruit ce fichier temporaire après avoir recopier son contenu dans le fichier initial. Cette procédure appelle la procédure chat ( ) pour informer tous les clients de la déconnexion des chateurs.

CODE :

(Voir Le code source si joints)

FONCTION 4:

chat_list(Client client, int actual)

DESCRIPTION :

Appelée par la procédure chat_join pour envoyer la liste des chateurs au client après l'acceptation de son nickname.

CODE :

(Voir Le code source si joints)

FONCTION 5:

clear_clients(Client *clients, int actual)

DESCRIPTION :

Fonction qui ferme les sockets Client pour terminer la session serveur correctement l'hors d'un arrêt.

CODE :

(Voir Le code source si joints)

FONCTION 6:

chat_join(Client client, int actual)

DESCRIPTION :

La procédure chat_join prend en paramètres le User :

* Elle consulte ensuite la liste des chateurs en ligne qui est enregistrée dans un fichier.
* Deux cas sont envisageables, si le Nick Name existe dans cette liste le serveur envoie au client
* une invitation à saisir un autre Nick Name sinon l'application enregistre le nouveau chateur en
* ajoutant son Nick Name à la fin du fichier et en lui envoyant un message de bienvenue dans le salon

* de chat et la liste des chateurs en ligne. Dans les deux cas le serveur envoie la réponse vers la
* socket du client .L'adresse du client est récupérée à partir de sa socket d'émission.

CODE :

(Voir Le code source si joints)

FONCTION 7:

write_client(SOCKET sock, const char *Buffer)

DESCRIPTION:

Cette fonction permet l'écriture dans la socket Client

CODE :

(Voir Le code source si joints)

FONCTION 8:

read_client(SOCKET sock, char *Buffer)

DESCRIPTION :

Fonction de Lecture depuis la socket Client

CODE :

(Voir Le code source si joints)

FONCTION 9:

init_connection(const char *address, const char *PORT)

DESCRIPTION :

Fonction qui initialise la connexion de la socket Client

CODE :

(Voir Le code source si joints)

FONCTION 10:

end_connection(int sock)

DESCRIPTION :

Fonction de la socket Serveur.

CODE :

(Voir Le code source si joints)

FONCTION 11:

chat_ Room(const char *address, const char *PORT)

DESCRIPTION:

Fonction principal de traitement du serveur elle englobe toutes les opérations qui serons effectué l'hors du chat, le code est commenté et bien décrit chaque fonction seras précédé d'un commentaire qui l'expliquera.

CODE :

(Voir Le code source si joints)

LEXIQUE

FWRITE

```
size_t fwrite(const *ptr, size_t size, size_t nmemb, FILE *stream);
```

Fonction, écrit dans stream nmemb blocs chacun de taille size lus dans la mémoire pointée par ptr.
Rend le nombre d'éléments effectivement écrits.

FCLOSE

```
int fclose(FILE *);
```

Fonction, ferme un fichier.
Rend 0 si succès, EOF sinon

FSEEK

```
int fseek(FILE *stream, long offset, int whence);
```

Fonction, positionne stream
- A la position offset si whence vaut SEEK_SET.
- A la position courante + offset si whence vaut SEEK_CUR
- à la fin du fichier + offset si whence vaut SEEK_END (pas forcément implémenté)

Rend 0 si succès, autre chose sinon.

FTELL

```
long ftell(FILE *);
```

Fonction de consultation.
Rend la position courante du fichier.

REWIND

```
void rewind(FILE *);
```

Fonction, met la position courante du fichier au début.

FGETS

```
char *fgets(char *s, int n, FILE *stream);
```

Fonction, charge s (qui doit être allouée) avec une chaîne de caractères pris dans stream. Lit jusqu'au premier \n ou fin-de-fichier, mais pas plus de n caractères.
Un caractère nul est mis à la fin de la chaîne.
Rend s.

STRFTIME

```
size_t strftime(char *s, size_t maxsize, const char * format, const struct tm *
timeptr);
```

Fonction, écrit dans s une chaîne de caractères maîtrisée par format représentant le temps pointé par timeptr.
Le format comprend une liste de switchs spécifiques.
Rend le nombre de caractères écrits.

MEMMOVE

```
void *memmove(void * s1, const void * s2, size_t n);
```

La fonction memmove() déplace le contenu d'une zone mémoire vers une autre. Elle prend en argument le pointeur vers la zone mémoire allouée vers laquelle déplacer le contenu mémoire (dest), le pointeur vers la zone de départ de la copie (src) et le nombre d'octets à déplacer. Elle retourne le pointeur vers la zone de destination (dest).

Contrairement à memcpy(), il est possible que les 2 zones se supperposent.

ACCEPT

int accept(int *sock*, **struct sockaddr** **adresse*, **socklent_t** **longueur*);
accept est utilisé généralement avec des processus serveurs orientés-connexion.
L'argument *sock* est une socket qui a été créée avec la fonction **socket**(2) . On lui a affecté une adresse avec **bind**(2) . Enfin on a indiqué au système, avec **listen**(2) qu'elle désirait recevoir des connexions entrantes.
L'appel système **accept** extrait la première connexion de la file des connexions en attente, crée une nouvelle socket avec essentiellement les mêmes propriétés que *sock* et alloue un nouveau descripteur de fichier pour cette socket.

WSADATA

Le WSAData la structure contient des informations sur l'implémentation Windows Sockets.

SYNTAXE

```
typedef struct {WSAData
  WVersion WORD;
  WHighVersion WORD;
  szDescription char [WSADESCRIPTION_LEN +1];
  omble szSystemStatus [WSASYS_STATUS_LEN +1];
  non signés iMaxSockets courts;
  unsigned iMaxUdpDg courte;
  omble FAR * lpVendorInfo;
WSAData}, * LPWSADATA;
```

FD_SET

Le fd_set structure est utilisée par Windows Sockets diverses fonctions et les fournisseurs de services, tels que la sélection de fonction, pour placer prises dans un "set" à des fins diverses, telles que l'essai d'un socket donné pour une meilleure lisibilité à l'aide de la *READFDS* paramètre de la sélection de fonction.

SYNTAXE

```
fd_set typedef struct {
  u_int fd_count;
  PRISE fd_array [FD_SETSIZE];
Fd_set}
```

FD_ZERO

Initialiser le descripteur de fichier set

Cette fonction initialise le descripteur de fichier mis à ne contiennent pas de descripteurs de fichiers.

Considérations de programmation

Si vous voulez surveiller plus de 256 descripteurs de fichiers sur un seul (tpf_select_bsd) appel, l'application doit définir sa propre FD_SETSIZE avant la comprennent de la tête (sys / time.h.) Le but de ces fonctions FD est de mettre en place l'entrée et vérifier l'(sortie) d'un appel (tpf_select_bsd). Le FD_CLR, FD_COPY, FD_SET, et les fonctions FD_ZERO (ensemble) jusqu'à l'entrée de l'appel (tpf_select_bsd). La fonction FD_ISSET vérifie la sortie de l'appel (tpf_select_bsd).

FD_ISSET

FD_ISSET() renvoie une valeur non nulle si un descripteur de fichier indiqué est présent dans un ensemble et zéro s'il ne l'est pas.

```
if (fd2 > 0)
    if (FD_ISSET(fd2, &wr)) {
         r = write(fd2, buf1 + buf1_written,
                     buf1_avail - buf1_written);
                       if (r < 1)
                           SHUT_FD2;
                       else
                           buf1_written += r;
     }
```

BIBLIOGRAPHIE

http://www.siteduzero.com/tutoriel-3-3279-manipulation-de-sockets.html
https://www.rocq.inria.fr/secret/Anne.Canteaut/COURS_C/chapitre6.html
http://doc.ubuntu-fr.org/zenity
http://ascii-table.com/ansi-escape-sequences.php
http://www.areaprog.com/c/cours-232-fread-et-fgets-lecture-dans-un-fichier