TP Final - Sockets et HTTP

Partie I: Socket

1. Configuration

Le fichier tp_5.c configure une socket qui va attendre une connection sur un port et une addresse définie, afficher un message dans la console quand elle recoit une tentative de connection et s'arrêter.

1. Lisez le code puis compilez le. Lancez le et vérifiez qu'il accepte bien une connection à l'addresse http://127.0.0.1:7070/ via un navigateur ou cURL (Attention, comme le programme ne renvoie pas de réponse, votre navigateur vous donnera une erreur. Pour vérifier que le programme fonctionne, regardez le terminal).
2. Créez une structure qui contiendra toutes les informations relatives à la configuration de la socket (addresses acceptées, protocole ipv4 ou 6, TCP ou UDP, port d'écoute, nombre maximum de connections en attente). Cette structure pourra contenir un champ de type sockaddr_in.
3. Créez une fonction nommée « run_server » qui prend en argument la structure précédement créée et qui
    - Crée la socket
    - Attache la socket à l'addresse précisée
    - Demande à la socket d'écouter les connections.
    - Accepte une connection et affiche le numéro de la socket créée
4. Modifiez la fonction run_server pour qu'elle accepte plusieurs connections. À chaque connection, elle affichera le numéro de la socket créée dans la console puis elle la fermera.

2. Lire et Ecrire

Pour lire les informations envoyées dans la socket nous utilisons la fonction `read` sur la socket créée pour communiquer avec le client.

La fonction read accepte en argument :

- le descripteur de fichier de la socket créée pour ce client (**attention !** Il s'agit de l'identifiant renvoyé par la fonction accept()).
- un tableau de charactères dans lequel elle écrira le message reçu.
- un entier qui indique le nombre d'octets a lire.

1. Modifiez votre code pour qu'il lise les 1024 premiers octects reçus et les affiche dans la console.
2. Modifiez la fonction pour aussi renvoyer une chaine de charactères au client. Pour cela utilisez la fonction `send` qui accepte en paramètres :
    - Le descripteur de fichier de la socket sur laquelle écrire.
    - un tableau de charactères contenant le message à envoyer.
    - un entier qui indique le nombre d'octets à envoyer.
    - un entier contenant des flags (ici, on passera 0).

Si vous vous connectez depuis un navigateur vous devriez voir le résultat renvoyé

Partie 2. HTTP

3. Requête

Pour échanger avec un navigateur, nous utilisons le protocole HTTP. Pour cela nous devons attendre une requête d'un format précis et renvoyer une réponse dans un autre format.

Une requête HTTP est composée de trois élements, la première ligne (parfois appelée request line en anglais), les options et un message (optionnel).

Pour l'instant nous nous concentrerons sur la première ligne, elle contient 3 elements séparés par des espaces :

- Le verbe (un mot qui indique le type de requête), pour l'instant nous n'utiliserons que le verbe GET.
- L'url.
- La version du protocole HTTP.

L'url est au format suivant :

addresse/route/?params

- Dans ce TP, l'addresse est localhost:7070.
- La route sera simplement un mot.
- Les paramètres sont des couples clé=valeur séparés par des « & ».

Par exemple on pourrait avoir l'url suivante :

http://localhost:7070/addUser/?name=bob&age=23

1. Aidez vous de la fonction strtok() pour séparer les trois informations de la première ligne (verbe, url, version du protocole) puis affichez les sur 3 lignes différentes dans la console.
2. Aidez vous toujours de la fonction strtok pour séparer l'addresse, la route et les paramètres. Affichez ces informations dans la console pour vérifier que votre programme fonctionne correctement.
3. Enregistrez les informations de la requête dans une structure qui contiendra:
    - Une chaine de charactères contenant la route.
    - Une liste de structures contenant les couples clé-valeur.

4. Réponse

Jusqu'à présent, nous renvoyons du texte arbitraire au navigateur. Ce n'est pas une réponse http correcte mais le navigateur l'affiche quand même.

Une réponse http est composée de :

- Une première ligne.
- Une serie d'options sur des lignes différentes.
- Une ligne vide.
- Un message sur plusieurs lignes.

Note: Dans ce TP nous utiliserons uniquement les status 200 et 404. Lorsque le status est 404 nous devons écrire 404 NOT FOUND. Lorsque le status est 200 nous écrivons 200 OK.

1. Modifiez votre code pour renvoyer la réponse suivante :

    ```
    "HTTP/1.1 200 OK
    Content-Type : html

    <h1>réponse</h1>
    <p>Bonjour</p>"
    ```

    Testez votre code dans un navigateur

2. Modifiez votre code pour qu'il renvoie cette réponse uniquement si la route est "home" et qu'il renvoie une erreur 404 si la route est inconnue.

Partie 3. Dessin

5. Statique

Pour cet exercice vous aurez besoin de la fonction read_file ci dessous. Cette fonction accepte en argument un nom de fichier et renvoie un tableau de charactères contenant le contenu du fichier.

La mémoire renvoyée par read_file doit être libérée avec free après utilisation.

```
char* read_file(char* path){
    char * buffer;
    FILE * f = fopen (path, "rb");

    if (f)
    {
       // déplace le curseur jusqu'au charactère représentant la fin du fichier
        fseek (f, 0, SEEK_END);
        long length = ftell (f); // ftell renvoie la position du curseur
        buffer = malloc (length + 1);// alloue de la mémoire égale à la taille du fichier
        fseek (f, 0, SEEK_SET);  // redéplace le curseur au début du fichier
        if (buffer)
        {
            fread (buffer, 1, length, f);
        }
        buffer[length] = 0;
        fclose (f);
    } else {
        printf("file not oppened : %s\n", path);
    }
    return buffer;
}
```

1. Utilisez la fonction read_file ci dessous pour renvoyer le contenu du fichier "home.html" en tant que message lorsque la route est "home" (Vous pouvez uiliser la fonction concat du tp précédent pour concaténer le header de la requête avec le contenu).
2. Créez une fonction qui sera appelée quand la route est "get_pixels" et qui renverra le texte suivant.

   ```
   Content-Type: application/json
   Access-Control-Allow-Origin: *

   [
       {"x": 5, "y": 10, "color": "#ff0000"},
       {"x": 15, "y": 20, "color": "#00ff00"},
       {"x": 25, "y": 30, "color": "#0000ff"}
   ]
   ```

   Vérifiez que lorsque vous vous connectez dans le navigateur, trois pixels de couleurs différentes sont affichés sur votre écran.
3. Créez une structure pixel contenant une position x et y ainsi qu'une couleur. Créez un tableau de pixels et utilisez le pour modifier la réponse envoyée précédement (vous pouvez utiliser les fonctions de concaténation du TP 4 ainsi que la fonction itoa pour convertir un nombre en chaine de charactères).

   La fonction sprintf fonctionne comme printf mais accepte en premier argument une chaine de charactères. Au lieu d'être ecrit dans la console, le texte sera ecrit dans la chaine de charactères passée en argument.

6. Dynamique

1. Utilisez le code du tp précédent (Exercice: Vecteur) pour créer une liste de pixels de taille dynamique.
2. Modifiez le code pour qu'un appel à la route "add_pixel" avec "x", "y" et "color" en paramètres ajoute un pixel dans le tableau (Utilisez la fonction atoi pour convertir des chaines de charactères en entiers).
3. Vérifiez que plusieurs utilisateurs peuvent se connecter en même temps et dessiner.

Partie 4. Client

7. Dessin depuis le terminal

Ci dessous un code qui ouvre une socket C et envoie un message au serveur

```
char* req = "GET http:127.0.0.1/home/ HTTP/1.1";

struct sockaddr_in address;
address.sin_family = AF_INET;
address.sin_port = htons(7070);
inet_pton(AF_INET, "127.0.0.1", &address.sin_addr);

// Compléter ici :
int socket_id = //;
connect(socket_id, (struct sockaddr*)&address, sizeof(address));


// Compléter ici en utilisant send et read


close(socket_id);
return 0;
```

1. Complétez ce code pour qu'il affiche dans la console la réponse du serveur, vous devriez obtenir le contenu de la page html.
2. Modifiez ce code pour qu'il envoie des pixels au serveur afin de dessiner un cercle.
3. Modifiez le code pour qu'il accepte en argument la position du cercle, sa couleur et son rayon.

8. Épilogue

Bravo ! Vous avez complétés tous les TP.

Vérifiez que votre code ne contient pas de bugs ou de failles de sécurités ou entrainez vous au contrôle avec le sujet prévu à cet effet.