My Project

Generated by Doxygen 1.8.8

Fri Dec 11 2015 09:44:04

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 message Struct Reference

Objet message qui peut etre soit une requete, soit une reponse. utilisé pour communiquer entre le programme et la bibliothèque.

`#include <request.h>`

Data Fields

- char * **buf**
- unsigned int **len**
- unsigned int **clientId**
- struct sockaddr_in * **clientAddress**

3.1.1 Detailed Description

Objet message qui peut etre soit une requete, soit une reponse. utilisé pour communiquer entre le programme et la bibliothèque.

Pour les requetes l'allocation du pointeur buf est fait par la librairie, le programme recoit l'information par la fonction getRequest, et une fois le message traité par le programme, il doit libérer explicitement le message par la fonction freeRequest.

Pour les reponses, la librairie recopie les données pointées dans une structure de message interne, le programme peut donc liberer la memoire si necessaire tout de suite après l'appel à sendReponse.

3.1.2 Field Documentation

3.1.2.1 char* buf

buf Un pointeur vers le message recu

3.1.2.2 struct sockaddr_in* clientAddress

iclientAddress pointeur vers une structure permettant de recuperer l'adresse IP et le port du client HTTP

3.1.2.3 unsigned int clientId

clientId identifiant du client, recu dans une requete, doit etre recopier dans la reponse correspondante

3.1.2.4 unsigned int len

len La longueur du message recu

The documentation for this struct was generated from the following file:

- api/**request.h**

Chapter 4

File Documentation

4.1 api/request.h File Reference

Interface de traitement des requetes et reponses HTTP.

Data Structures

- struct **message**

  Objet message qui peut etre soit une requete, soit une reponse. utilisé pour communiquer entre le programme et la bibliothèque.

Macros

- #define **MAXCLIENT** 10

Functions

- **message** ∗ **getRequest** (short int port)

  Fonction de recuperation d'une requete, cette fonction est bloquante et doit etre appeler dans la boucle principale du programme. Cette fonction essaie de recuperer une requete entière pour la livrer à votre programme, mais si des cas d'ereur se produise, livre ce qui a été recu à l'instant t sans filtrage, c'est votre programme qui devra traiter ces cas d'erreurs.
- void **freeRequest** (**message** ∗r)

  Procedure de liberation de la memoire quand le programme en a fini avec une requete (message ∗).
- void **sendReponse** (**message** ∗r)

  Procedure d'envoi d'un mesage au client HTTP.
- void **requestShutdownSocket** (int i)

  Procedure de demande de cloture de la connexion, si la bibliothèque à encore des données à envoyer d'un send↩ Reponse précédent, la connexion ne sera fermée qu'à la fin de cet envoi.

4.1.1 Detailed Description

Interface de traitement des requetes et reponses HTTP.

Author

Quentin Giorgi

Version

1.0

Date

13 Decembre 2015

Fichier d'interface entre la bibliothèque et votre programme. Votre programme doit inclure ce fichier d'entete #include <**request.h** (p. 7)> La compilation doit inclure l'option -L. -lrequest

Exemple de programme:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "request.h"

int main(int argc, char *argv[])
{
    message *requete;
    message *reponse;

    while ( 1 ) {
        // on attend la reception d'une requete
        requete=getRequest(8080);

        // Affichage de debug
        printf("########################################\nDemande recue depuis le client %d\n",requete->
      clientId);
        printf("Client [%d] [%s:%d]\n",requete->clientId,inet_ntoa(requete->
      clientAddress->sin_addr),htons(requete->clientAddress->sin_port));
        printf("Contenu de la demande %.*s\n\n",requete->len,requete->buf);

        // Si on a une reponse a faire
        if (argv[1]) {
            if (reponse=malloc(sizeof(message))) {
                reponse->buf=argv[1]; // on devrait l'allouer sinon
                reponse->len=strlen(argv[1]);
                reponse->clientId=requete->clientId;
                sendReponse(reponse);
                // reponse est recopiée on peut tout de suite liberer la memoire
                free(reponse);
                //optionnel, ici on clot la connexion tout de suite (HTTP/1.0)
                requestShutdownSocket(reponse->clientId);
            }
        }
        // on ne se sert plus de requete a partir de maintenant, on peut donc liberer...
        freeRequest(requete);
    }

    return (1);
}
```

4.1.2 Function Documentation

4.1.2.1 void freeRequest (message * r)

Procedure de liberation de la memoire quand le programme en a fini avec une requete (message *).

Parameters

r	Le message à libérer.

4.1.2.2 message * getRequest (short int *port*)

Fonction de recuperation d'une requete, cette fonction est bloquante et doit etre appeler dans la boucle principale du programme. Cette fonction essaie de recuperer une requete entière pour la livrer à votre programme, mais si des cas d'ereur se produise, livre ce qui a été recu à l'instant t sans filtrage, c'est votre programme qui devra traiter ces cas d'erreurs.

Parameters

port	port d'ecoute de la socket, utilisé qu'au premier appel de la fonction, ensuite ce parametre est ignoré dans les appels successifs.

Returns

un pointeur vers une structure message.

4.1.2.3 void requestShutdownSocket (int *i*)

Procedure de demande de cloture de la connexion, si la bibliothèque à encore des données à envoyer d'un send↩ Reponse précédent, la connexion ne sera fermée qu'à la fin de cet envoi.

Parameters

i	L'Id du client dont on doit fermer la connexion.

4.1.2.4 void sendReponse (message * *r*)

Procedure d'envoi d'un mesage au client HTTP.

Parameters

r	Le message à envoyer (recopié par la bibliothèque)

Index