Compte Rendu TD 2 : Programmation C et Réseau

Aïman Al-SABAGH-NAHHAS et Alain PIALLAT

11 avril 2025

Présentation du projet

Système de boite aux lettres

Le système de boîte aux lettres repose sur un serveur appelé boîte aux lettres qui va stocker les messages envoyé par les différents émeteurs. Les émetteurs vont se connecté au serveur pour envoyer des messages qui seront stocké dans les boîtes aux lettres des recepteurs, qui se connecteront au serveur pour récupérer les messages qui sont dans leur boite aux lettres.

Communication entre l'émetteur et le serveur

1 - L'émetteur crée la connexion avec le serveur en TCP.
2 - Le serveur crée un socket dédié à la communication avec le client.
3 - Le client envoie un message au serveur contenant :

- le caractère 'E' pour indiquer qu'il s'agit d'un émetteur,
- la longueur des messages qui sont envoyées au serveur,
- le nombre de messages qui sont envoyées au serveur.

4 - Le client envoie ensuite les messages au serveur.
5 - Le serveur stocke les messages dans le système de boîte aux lettres.

Communication entre le serveur et le recepteur

1 - Le recepteur se connecte au serveur en TCP.
2 - Le serveur crée un socket dédié à la communication avec le client.
3 - Le client envoie un message au serveur contenant :

- le caractère 'R' pour indiquer qu'il s'agit d'un recepteur,
- le numéro de la boite aux lettres du recepteur.

4 - Le serveur envoie au recepteur les messages qui sont dans sa boite aux lettres. Et vide au fur et à mesure la boite aux lettres du recepteur.
5 - Le serveur ferme la connexion avec le recepteur quand la boite aux lettres est vide.

Gestion du système de boite aux lettres

La structure de données choisi pour stocker les messages est une liste chaînée de boites aux lettres. Chaque boite aux lettres est une liste chaînée de messages.

Implémentation en C des structures de données

```c
typedef struct message {
    char *message;
    int length;
    struct message *next;
} message;
typedef struct mailbox {
```

```
    int number;
    struct message *messages;
    struct mailbox *next;
} mailbox;
```

Implémentation du système de boite aux lettres

get_mailbox

Récupère la boite aux lettres correspondant au numéro fourni en paramètre. Si la boite aux lettres n'existe pas, elle est créée. La fonction renvoie un pointeur vers la boite aux lettres.

Signature en C :

```
mailbox *get_mailbox(mailbox **mailboxes, int number);
```

Paramètre	Type	Description
mailboxes	mailbox **	Pointeur vers la liste chaînée de boites aux lettres.
number	int	Numéro de la boite aux lettres à récupérer.
Retour	mailbox *	Pointeur vers la boite aux lettres.

Implémentation en pseudocode :

```
function get_mailbox
    input: mailboxes, number
    output: mailbox

    current = mailboxes
    while current != NULL do
        if current->number == number then
            return current
        end if
        current = current->next
    end while

    new_mailbox = malloc(sizeof(mailbox))
    new_mailbox->number = number
    new_mailbox->messages = NULL
    new_mailbox->next = mailboxes
    mailboxes = new_mailbox

    return new_mailbox
end function
```

add_message

Ajoute un message à la boite aux lettres fournie en paramètre. La fonction renvoie 0 si l'ajout a réussi, -1 sinon.

Signature en C :

```c
int add_message(mailbox *mail_box, char *message, int length);
```

Paramètre	Type	Description
mail_box	mailbox *	Pointeur vers la boite aux lettres.
message	char *	Message à ajouter à la boite aux lettres.
length	int	Longueur du message.
Retour	int	0 si l'ajout a réussi, -1 sinon.

Implémentation en pseudocode :

```
function add_message
    input: mail_box, message, length
    output: int

    new_message = malloc(sizeof(message))
    new_message->message = malloc(length)
    memcpy(new_message->message, message, length)
    new_message->length = length
    new_message->next = mail_box->messages
    mail_box->messages = new_message

    return 0
end function
```

get_message

Récupère le message suivant de la boite aux lettres fournie en paramètre. La fonction renvoie un pointeur vers le message. Si la boite aux lettres est vide, la fonction renvoie NULL. La fonction libère la mémoire du message récupéré.

Signature en C :

```c
char *get_message(mailbox *mail_box, int *length);
```

Paramètre	Type	Description

Paramètre	Type	Description
mail_box	mailbox *	Pointeur vers la boite aux lettres.
length	int *	Pointeur vers la longueur du message.
Retour	char *	Pointeur vers le message.

Implémentation en pseudocode :

```
function get_message
    input: mail_box, length
    output: char *

    if mail_box->messages == NULL then
        return NULL
    end if

    message = mail_box->messages
    mail_box->messages = mail_box->messages->next
    length = message->length

    content = malloc(length)
    memcpy(content, message->message, length)
    free(message->message)
    free(message)
    return content
end function
```

free_mailboxes

Libère la mémoire allouée pour les boites aux lettres. La fonction libère la mémoire de chaque boite aux lettres et de chaque message contenu dans la boite aux lettres.

Signature en C :

```c
void free_mailboxes(mailbox **mailboxes);
```

Paramètre	Type	Description
mailboxes	mailbox **	Pointeur vers la liste chaînée de boites aux lettres.
Retour	void	

Implémentation en pseudocode :

```
function free_mailboxes
    input: mailboxes
```

```
    output: void

    current = mailboxes
    while current != NULL do
        next = current->next
        while get_message(current->messages, NULL) != NULL do
        end while
        free(current)
        current = next
    end while
end function
```

Implémentation de l'émetteur

emitter

L'émetteur est un programme qui va se connecter au serveur pour envoyer des messages. L'émetteur va créer une connexion TCP avec le serveur, envoyer les messages au serveur et fermer la connexion.

Signature en C :

```c
int emitter(char *hostname, int port, int length, int count, int number);
```

Paramètre	Type	Description
hostname	char *	Nom d'hôte du serveur.
port	int	Port du serveur.
length	int	Longueur des messages à envoyer.
count	int	Nombre de messages à envoyer.
number	int	Numéro de la boite aux lettres destinataire.
Retour	int	0 si l'émetteur a réussi, -1 sinon.

Implémentation en pseudocode :

```
function emitter
    input: hostname, port, length, count
    output: int

    // Créer un socket TCP
    sock <- socket(AF_INET, SOCK_STREAM, 0)

    // Construire l'adresse du serveur
    server_address <- constuct_address(hostname, port)
```

```
    // Se connecter au serveur
    while connect(sock, server_address) != 0 do
        sleep(1)
    end while

    // Envoyer le message d'initialisation
    message <- "E" + length + count + number
    // Le message est de la forme "E<length><count><number>"
    send(sock, message, sizeof(message), 0)

    // Envoyer les messages
    for i from 0 to count do
        message <- generate_message(length)
        send(sock, message, length, 0)
    end for

    // Fermer la connexion
    shutdown(sock, 2)
    close(sock)
    return 0
end function
```

Implémentation du recepteur

receiver

Le recepteur est un programme qui va se connecter au serveur pour récupérer les messages. Le recepteur va créer une connexion TCP avec le serveur, envoyer le numéro de la boite aux lettres et récupérer les messages.

Signature en C :

```c
int receiver(char *hostname, int port, int number);
```

Paramètre	Type	Description
hostname	char *	Nom d'hôte du serveur.
port	int	Port du serveur.
number	int	Numéro de la boite aux lettres.
Retour	int	0 si le recepteur a réussi, -1 sinon.

Implémentation en pseudocode :

```
function receiver
    input: hostname, port, number
    output: int
```

```
    // Créer un socket TCP
    sock <- socket(AF_INET, SOCK_STREAM, 0)

    // Construire l'adresse du serveur
    server_address <- constuct_address(hostname, port)

    // Se connecter au serveur
    while connect(sock, server_address) != 0 do
        sleep(1)
    end while

    // Envoyer le message d'initialisation
    message <- "R" + number
    send(sock, message, sizeof(message), 0)

    // Recevoir les messages
    while true do
        message <- receive_message(sock)
        if message == NULL then
            break
        end if
        print(message)
        free(message)
    end while

    // Fermer la connexion
    shutdown(sock, 2)
    close(sock)
    return 0
end function
```

Implémentation du serveur

server_sender

Cette fonction est appelée par le serveur quand une connexion est établie avec un émetteur. La fonction va récupérer les messages envoyés par l'émetteur dans le socket qui lui est dédié et les ajouter à la boite aux lettres. La fonction va ensuite fermer le socket une fois que tous les messages ont été reçus.

Signature en C :

```c
void *server_sender(int sock, mailbox *mbox, int length, int count);
```

Paramètre	Type	Description
sock	int	Socket dédié à la communication avec l'émetteur.
mbox	mailbox *	Pointeur vers la boite aux lettres destinataire.

Paramètre	Type	Description
length	int	Longueur des messages reçus.
count	int	Nombre de messages reçus.
Retour	void	

Implémentation en pseudocode :

```
function server_sender
    input: sock, mailboxes, length, count
    output: void

    // Recevoir les messages
    for i from 0 to count do
        message <- receive_message(sock)
        if message == NULL then
            break
        end if
        add_message(mbox, message, length)
        free(message)
    end for

    // Fermer le socket
    shutdown(sock, 2)
    close(sock)
    return 0
end function
```

server_receiver

Cette fonction est appelée par le serveur quand une connexion est établie avec un recepteur. La fonction va récupérer les messages de la boite aux lettres et les envoyer au recepteur. La fonction va ensuite fermer le socket une fois que tous les messages ont été envoyés.

Signature en C :

```c
void *server_receiver(int sock, mailbox *mbox);
```

Paramètre	Type	Description
sock	int	Socket dédié à la communication avec le recepteur.
mbox	mailbox *	Pointeur vers la boite aux lettres du recepteur.
Retour	void	

Implémentation en pseudocode :

```
function server_receiver
    input: sock, mbox
    output: void

    // Envoyer les messages
    while true do
        message <- get_message(mbox, NULL)
        if message == NULL then
            break
        end if
        send(sock, message, sizeof(message), 0)
        free(message)
    end while

    // Fermer le socket
    shutdown(sock, 2)
    close(sock)
    return 0
end function
```

server

Le serveur est le programme principal qui va gérer les connexions avec les émetteurs et les recepteurs. Le serveur va créer un socket d'écoute pour accepter les connexions, créer un thread pour chaque connexion et gérer la liste chaînée de boites aux lettres.

Signature en C :

```c
int server(int port);
```

Paramètre	Type	Description
port	int	Port d'écoute du serveur.
Retour	int	0 si le serveur a réussi, -1 sinon.

Implémentation en pseudocode :

```
function server
    input: port
    output: int

    // Créer un socket d'écoute
    sock <- socket(AF_INET, SOCK_STREAM, 0)

    // Lier le socket à l'adresse du serveur
    server_address <- construct_address(INADDR_ANY, port)
    bind(sock, server_address, taille_address)
```

```
    // Écouter les connexions entrantes
    listen(sock, 5)

    // Accepter les connexions entrantes
    while true do
        client_sock <- accept(sock, address_client, taille_address_client)
        if client_sock == -1 then
            continue
        end if

        // récupérer le message d'initialisation
        message <- receive_message(client_sock)
        if message == NULL then
            close(client_sock)
            continue
        end if
        if message[0] == 'E' then
            length <- message[1]
            count <- message[2]
            number <- message[3]
            mbox <- get_mailbox(mailboxes, number)
            thread_create(server_sender, client_sock, mbox, length, count)
        else if message[0] == 'R' then
            number <- message[1]
            mbox <- get_mailbox(mailboxes, number)
            thread_create(server_receiver, client_sock, mbox)
        end if
        // Libérer la mémoire du message
        free(message)
    end while

    // Libérer la mémoire des boites aux lettres
    free_mailboxes(mailboxes)

    // Fermer le socket d'écoute
    close(sock)
    return 0
end function
```