TRAVAUX PRATIQUES

Architectures des réseaux

Travail réalisé par :

Yasser Chouket

Youssef Chekili

Niveau:

GL2

Année Universitaire :

2024-2025

I. Objectifs

Les objectifs de ce TP sont :

- Comprendre et utiliser l'interface *sockets* pour les communications réseau.
- Développer un client HTTP fonctionnel.
- Mettre en œuvre une application Client/Serveur en mode connecté (TCP).
- Implémenter une version en mode non connecté (UDP).
- Concevoir un serveur concurrent capable de répondre à plusieurs clients et services en parallèle

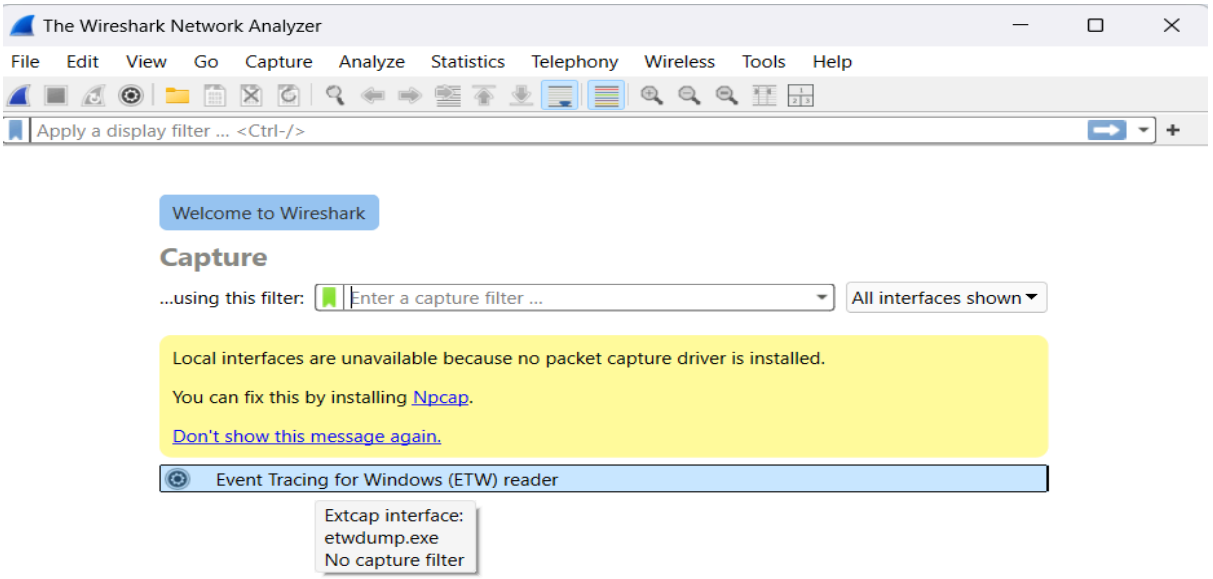
II. Outils de capture de paquets

a. Utilisation de tcpdump

L'outil tcpdump permet de capturer les paquets circulant sur une interface réseau donnée.

b. Utilisation de Wireshark

Wireshark propose une interface graphique pour visualiser les paquets. Il permet aussi d'appliquer des filtres très précis.



III. Développement d'un client HTTP

1. Vérification de la présence du serveur HTTP

Pour vérifier qu'un serveur HTTP est en fonctionnement, il faut envoyer des requêtes et attendre l'établissement d'une connexion, par exemple en utilisant la commande `telnet` suivie de l'adresse IP du serveur et du port (généralement 80 pour un serveur HTTP).

```
bor3i@yasser:~$ telnet 10.250.101.1 80
Trying 10.250.101.1...
```

➔ **Envoi d'une requête avec la méthode HEAD**

```
bor3i@yasser:~$ telnet httpbin.org 80
Trying 54.236.92.255...
Connected to httpbin.org.
Escape character is '^]'.
HEAD / HTTP/1.1
Host: httpbin.org
Accept: */*

HTTP/1.1 200 OK
Date: Sat, 03 May 2025 19:03:49 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 9593
Connection: keep-alive
Server: gunicorn/19.9.0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
```

➔ **Envoi d'une requête avec la méthode GET**

```
bor3i@yasser:~$ telnet httpbin.org 80
Trying 18.205.89.57...
Connected to httpbin.org.
Escape character is '^]'.
GET / HTTP/1.1
Host: httpbin.org
Accept: */*

HTTP/1.1 200 OK
Date: Sat, 03 May 2025 19:06:19 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 9593
Connection: keep-alive
Server: gunicorn/19.9.0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true

<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>httpbin.org</title>
    <link href="https://fonts.googleapis.com/css?family=Open+Sans:400,700|Source+Code+Pro:300,600|Titillium+Web:400,600,700"
        rel="stylesheet">
    <link rel="stylesheet" type="text/css" href="/flasgger_static/swagger-ui.css">
```

on obtient le contenu de la réponse non seulement le header, dans ce cas c'est de type html.

2.Client http en mode connecté

On utilise les sockets pour créer un programme c qui va être un proxy entre nous et le serveur, ce programme va connecter au serveur http et attendre un saisis d'une requête dont il envoie vers le serveur et nous transmet la réponse, voici un prototype simple de ce programme :

```
joseph04@Ubuntu:~/tp_rizou$ nano client_http.c
joseph04@Ubuntu:~/tp_rizou$ nano client_http.c
joseph04@Ubuntu:~/tp_rizou$ gcc client_http.c -o client_http
joseph04@Ubuntu:~/tp_rizou$ ./client_http 142.250.74.196 80
Connexion au serveur HTTP 142.250.74.196:80...
Requête HTTP envoyée.
Réponse du serveur :

HTTP/1.1 301 Moved Permanently
Location: http://www.google.com/
Content-Type: text/html; charset=UTF-8
Content-Security-Policy-Report-Only: object-src 'none';base-uri 'self';script-src 'nonce--g
C1kByQrUywpu72eiQFAQ' 'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline' https:
 http:;report-uri https://csp.withgoogle.com/csp/gws/other-hp
Date: Fri, 02 May 2025 18:00:49 GMT
Expires: Sun, 01 Jun 2025 18:00:49 GMT
Cache-Control: public, max-age=2592000
Server: gws
Content-Length: 219
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Connection: close

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>

Connexion fermée.
```

```c
// client_http.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netdb.h>

#define BUFFER_SIZE 4096

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <adresse_ip> <port>\n", argv[0]);
        return 1;
    }

    const char *ip = argv[1];
    int port = atoi(argv[2]);

    int sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        perror("Erreur de socket");
        return 1;
    }

    struct sockaddr_in serv_addr;
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(port);

    if (inet_pton(AF_INET, ip, &serv_addr.sin_addr) <= 0) {
        perror("Adresse invalide");
        close(sock);
        return 1;
    }
```

```c
    char requete[] = "GET / HTTP/1.1\r\nHost: 142.250.74.196 \r\nConnection: close\r\n\r\n";
    send(sock, requete, strlen(requete), 0);
    printf("Requête HTTP envoyée.\n");

    char buffer[BUFFER_SIZE];
    int n;
    printf("Réponse du serveur :\n\n");
    while ((n = recv(sock, buffer, BUFFER_SIZE - 1, 0)) > 0) {
        buffer[n] = '\0';
        printf("%s", buffer);
    }

    printf("\nConnexion fermée.\n");
    close(sock);
    return 0;
}
```

On envoie une requête en utilisant le programme client :
On observe la requête HTTP envoyée par le client et capturé par tcpdump :

```
18:01:15.095464 IP 10.0.2.15.44974 > 142.250.74.196.80: Flags [.], ack 1, win 64
240, length 0
E..(rD@.@...
.....J....P}o.=..x.P.......
18:01:15.095624 IP 10.0.2.15.44974 > 142.250.74.196.80: Flags [P.], seq 1:61, ac
k 1, win 64240, length 60: HTTP: GET / HTTP/1.1
E..drE@.@...
.....J....P}o.=..x.P....#..GET / HTTP/1.1
Host: 142.250.74.196
Connection: close
```

IV. Transfert de message en mode connecté

Programme développé en C utilisant sockets pour envoyer une requête HTTP saiie par l'utilisateur.

➔ En implémentant tcp_client.c et tcp_server.c pour que le serveur envoie 60 fois des messages au client pour l'informer à propos du temps , on remarque bien que à chaque seconde , le server envoie via un packet le message temps pendant 60s càd 60 messages on étaient envoyés puis la fermeture de connexion commencée par le serveur

```
joseph04@Ubuntu:~/tp_rizou$ nano client_tcp.c
joseph04@Ubuntu:~/tp_rizou$ gcc client_tcp.c -o client_tcp
joseph04@Ubuntu:~/tp_rizou$ ./client_tcp
Connexion au serveur TCP...
Message 1 envoyé : Il est 18:09:09 !
Message 2 envoyé : Il est 18:09:10 !
Message 3 envoyé : Il est 18:09:11 !
Message 4 envoyé : Il est 18:09:12 !
Message 5 envoyé : Il est 18:09:13 !
Message 6 envoyé : Il est 18:09:14 !
Message 7 envoyé : Il est 18:09:15 !
Message 8 envoyé : Il est 18:09:16 !
Message 9 envoyé : Il est 18:09:17 !
Message 10 envoyé : Il est 18:09:18 !
```

```
joseph04@Ubuntu:~/tp_rizou$ nano client_http.c
joseph04@Ubuntu:~/tp_rizou$ nano serveur_tcp.c
joseph04@Ubuntu:~/tp_rizou$ gcc serveur_tcp.c -o serveur_tcp
joseph04@Ubuntu:~/tp_rizou$ ./serveur_tcp
Serveur prêt. En attente de connexion sur le port 12345...
Client connecté.
Message 1 reçu : Il est 18:09:09 !
Message 2 reçu : Il est 18:09:10 !
Message 3 reçu : Il est 18:09:11 !
Message 4 reçu : Il est 18:09:12 !
Message 5 reçu : Il est 18:09:13 !
Message 6 reçu : Il est 18:09:14 !
Message 7 reçu : Il est 18:09:15 !
Message 8 reçu : Il est 18:09:16 !
Message 9 reçu : Il est 18:09:17 !
Message 10 reçu : Il est 18:09:18 !
```

```
Message 50 reçu : Il est 18:09:58 !
Message 51 reçu : Il est 18:09:59 !
Message 52 reçu : Il est 18:10:00 !
Message 53 reçu : Il est 18:10:01 !
Message 54 reçu : Il est 18:10:02 !
Message 55 reçu : Il est 18:10:03 !
Message 56 reçu : Il est 18:10:04 !
Message 57 reçu : Il est 18:10:05 !
Message 58 reçu : Il est 18:10:06 !
Message 59 reçu : Il est 18:10:07 !
Message 60 reçu : Il est 18:10:08 !
Connexion fermée. Total messages reçus : 60
```

- On augmentant le temps d'attente tcp_server_delaied.c
  ->il n'y a pas de perte de messages

```
joseph04@Ubuntu:~/tp_rizou$ ./client_tcp
Connexion au serveur TCP...
Message 1 envoyé : Il est 18:29:39 !
Message 2 envoyé : Il est 18:29:40 !
Message 3 envoyé : Il est 18:29:41 !
Message 4 envoyé : Il est 18:29:42 !
Message 5 envoyé : Il est 18:29:43 !
Message 6 envoyé : Il est 18:29:44 !
Message 7 envoyé : Il est 18:29:45 !
Message 8 envoyé : Il est 18:29:46 !
Message 9 envoyé : Il est 18:29:47 !
Message 10 envoyé : Il est 18:29:48 !
```

```
Message 50 envoyé : Il est 18:30:29 !
Message 51 envoyé : Il est 18:30:30 !
Message 52 envoyé : Il est 18:30:31 !
Message 53 envoyé : Il est 18:30:32 !
Message 54 envoyé : Il est 18:30:33 !
Message 55 envoyé : Il est 18:30:34 !
Message 56 envoyé : Il est 18:30:35 !
Message 57 envoyé : Il est 18:30:36 !
Message 58 envoyé : Il est 18:30:37 !
Message 59 envoyé : Il est 18:30:38 !
Message 60 envoyé : Il est 18:30:39 !
Fin de l'envoi. Fermeture.
```

V. Transfert de données en mode non connecté

Maintenant , on travaille sur le protocole UDP au lieu de TCP . On a implémenté une version de serveur et client en UDP dans udp_server.c et udp_client.c

```
joseph04@Ubuntu:~/tp_rizou$ nano serveur_udp.c
joseph04@Ubuntu:~/tp_rizou$ gcc serveur_udp.c -o serveur_udp
joseph04@Ubuntu:~/tp_rizou$ ./serveur_udp
Serveur UDP en écoute sur le port 12346...
Message 1 reçu : UDP - Il est 18:17:03 !
Message 2 reçu : UDP - Il est 18:17:04 !
Message 3 reçu : UDP - Il est 18:17:05 !
Message 4 reçu : UDP - Il est 18:17:06 !
Message 5 reçu : UDP - Il est 18:17:07 !
Message 6 reçu : UDP - Il est 18:17:08 !
Message 7 reçu : UDP - Il est 18:17:09 !
Message 8 reçu : UDP - Il est 18:17:10 !
Message 9 reçu : UDP - Il est 18:17:11 !
Message 10 reçu : UDP - Il est 18:17:12 !
```

```
Message 50 reçu : UDP - Il est 18:17:52 !
Message 51 reçu : UDP - Il est 18:17:53 !
Message 52 reçu : UDP - Il est 18:17:54 !
Message 53 reçu : UDP - Il est 18:17:55 !
Message 54 reçu : UDP - Il est 18:17:56 !
Message 55 reçu : UDP - Il est 18:17:57 !
Message 56 reçu : UDP - Il est 18:17:58 !
Message 57 reçu : UDP - Il est 18:17:59 !
Message 58 reçu : UDP - Il est 18:18:00 !
Message 59 reçu : UDP - Il est 18:18:01 !
Message 60 reçu : UDP - Il est 18:18:02 !
Réception terminée. Total messages reçus : 60
```

->Tous les message sont bien reçus

- On augmentant le temps d'attente udp_server_delaied.c

```
joseph04@Ubuntu:~/tp_rizou$ nano serveur_tcp_delayed.c
joseph04@Ubuntu:~/tp_rizou$ nano serveur_udp_delayed.c
joseph04@Ubuntu:~/tp_rizou$ gcc serveur_udp_delayed.c -o serveur_udp_delayed
joseph04@Ubuntu:~/tp_rizou$ ./serveur_udp_delayed
Serveur UDP prêt sur le port 12346
Client connecté : 127.0.0.1:36042
Message 1 envoyé : Il est 18:34:24
Message 2 envoyé : Il est 18:34:26
Message 3 envoyé : Il est 18:34:28
Message 4 envoyé : Il est 18:34:30
Message 5 envoyé : Il est 18:34:32
Message 6 envoyé : Il est 18:34:34
Message 7 envoyé : Il est 18:34:36
Message 8 envoyé : Il est 18:34:38
Message 9 envoyé : Il est 18:34:40
Message 10 envoyé : Il est 18:34:42
```

VI. Serveur en mode concurrent

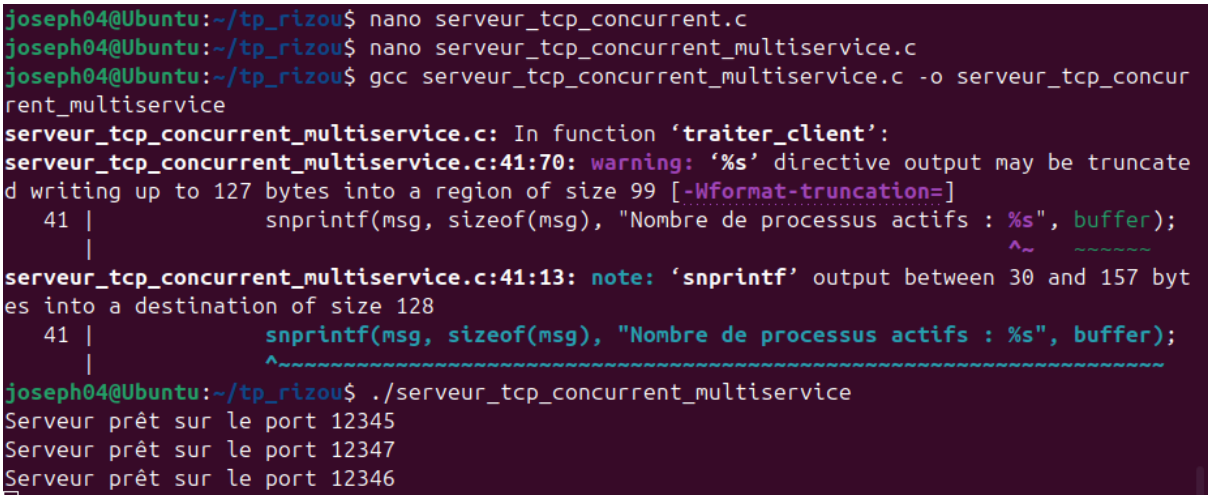
On modifie le code du serveur pour obtenir une sorte de serveur en mode concurrent , càd , un serveur qui fait fork() pour les différents processus (des exécutions) du code client et les manipulent simultanément . On observe bien que chaque fois qu'un client entre en communication avec le serveur via le 3-way handshake , le serveur le met en une sorte de file d'attente et à chaque fois envoie un paquet de données . Quand le serveur finit les 60 messages avec un certain client , il le retire de la file d'attente et passe au suivant

```
joseph04@Ubuntu:~/tp_rizou$ nano serveur_tcp_concurrent.c
joseph04@Ubuntu:~/tp_rizou$ gcc serveur_tcp_concurrent.c -o serveur_tcp_concurrent
joseph04@Ubuntu:~/tp_rizou$ ./serveur_tcp_concurrent
Serveur en attente de connexions sur le port 12345...
Client 1 connecté depuis 127.0.0.1:35724
Client 2 connecté depuis 127.0.0.1:38450
Client 3 connecté depuis 127.0.0.1:55128
```

```
joseph04@Ubuntu:~/tp_rizou$ ./client_tcp
Message 1 reçu : [Client 1] Message numéro 1

Message 2 reçu : [Client 1] Message numéro 2

Message 3 reçu : [Client 1] Message numéro 3

Message 4 reçu : [Client 1] Message numéro 4

Message 5 reçu : [Client 1] Message numéro 5

Message 6 reçu : [Client 1] Message numéro 6

Message 7 reçu : [Client 1] Message numéro 7

Message 8 reçu : [Client 1] Message numéro 8
```

```
joseph04@Ubuntu:~/tp_rizou$ ./client_tcp
Message 1 reçu : [Client 2] Message numéro 1

Message 2 reçu : [Client 2] Message numéro 2

Message 3 reçu : [Client 2] Message numéro 3

Message 4 reçu : [Client 2] Message numéro 4

Message 5 reçu : [Client 2] Message numéro 5

Message 6 reçu : [Client 2] Message numéro 6

Message 7 reçu : [Client 2] Message numéro 7

Message 8 reçu : [Client 2] Message numéro 8
```

- Si on ajoute plus de services aux serveurs on remarque que à chaque fois l'un des clients établit la connexion puis il attend que le client lui envoie en PSH+ACK le type de service demandé pour qu'il lui répond directement avec l'information appropriée et si il s'agit d'une demande d'affichage des 60 messages il le met dans la file de replys comme le cas de monoservice



```
joseph04@Ubuntu:~/tp_rizou$ nano serveur_tcp_concurrent.c
joseph04@Ubuntu:~/tp_rizou$ nano serveur_tcp_concurrent_multiservice.c
joseph04@Ubuntu:~/tp_rizou$ gcc serveur_tcp_concurrent_multiservice.c -o serveur_tcp_concur
rent_multiservice
serveur_tcp_concurrent_multiservice.c: In function ‘traiter_client’:
serveur_tcp_concurrent_multiservice.c:41:70: warning: ‘%s’ directive output may be truncate
d writing up to 127 bytes into a region of size 99 [-Wformat-truncation=]
   41 |             snprintf(msg, sizeof(msg), "Nombre de processus actifs : %s", buffer);
      |                                                                      ^~   ~~~~~~
serveur_tcp_concurrent_multiservice.c:41:13: note: ‘snprintf’ output between 30 and 157 byt
es into a destination of size 128
   41 |             snprintf(msg, sizeof(msg), "Nombre de processus actifs : %s", buffer);
      |             ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
joseph04@Ubuntu:~/tp_rizou$ ./serveur_tcp_concurrent_multiservice
Serveur prêt sur le port 12345
Serveur prêt sur le port 12347
Serveur prêt sur le port 12346
```

```
joseph04@Ubuntu:~/tp_rizou$ nano client_tcp_multiservice.c
joseph04@Ubuntu:~/tp_rizou$ gcc client_tcp_multiservice.c -o client_tcp_multiservice
joseph04@Ubuntu:~/tp_rizou$ ./client_tcp_multiservice 127.0.0.1 12345
Connecté au service sur le port 12345
Reçu : Il est 19:13:36

Reçu : Il est 19:13:37

Reçu : Il est 19:13:38

Reçu : Il est 19:13:39

Reçu : Il est 19:13:40

Reçu : Il est 19:13:41
```