MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

PROIECT

Protocoale de Internet

Client/Server C Program

Profesori Coordonatori:

Ing. Gabriel Lazăr

Prof.Dr.Ing Virgil-Mircea Dobrotă

Studenți:

Bulzan Alex-Călin

Marte Daniela

str. Memorandumului nr.28, 400114 Cluj-Napoca, România
tel. +40-264-401200, fax +40-264-592055, Dep. Relații Publice tel. +40-264-202284, +40-722-163346
http://www.utcluj.ro/

Prezentare

1. Obiectivul Proiectului:
   - Dezvoltarea unui program server/client în limbajul de programare C, sub sistemul de operare Linux, pentru a realiza comunicarea între programul ClientIPv4_win_2023.exe și resursele web IPv6 specificate.

2. Funcționalități Cheie:
   - a) Comunicare IPv4: Serverul Linux așteaptă cereri de la clienți la portul 22GSE, unde G=numărul grupei, S=numărul semigrupei, E=numărul echipei. Comunicarea se realizează prin socket stream IPv4. Socket Specific = 22311
   - b) Comunicare IPv6: Clientul IPv6 se conectează la URL-uri specifice conform comenzilor primite de la ClientIPv4_win_2023.exe.
   - c) Mesaj de Eroare: În cazul în care comanda nu este implementată, serverul Linux returnează un mesaj către clientul IPv4 Windows.
   - d) Comunicare HTTP IPv6: După stabilirea comunicării IPv6, clientul IPv6 trimite comanda GET / HTTP/1.0\r\n\r\n către Serverul IPv6 HTTP.

3. Tabelul de Asociere Comenzi - Destinații:

Comandă	Destinație
01#	www.yahoo.com
02#	www.6init.org
03#	www.ietf.org
...	...
20#	www.ja.net

4. Fluxul de Lucru:
   - Comunicare inițială prin socket stream IPv4 între ClientIPv4_win_2023.exe și serverul Linux.
   - Serverul gestionează comenzile și direcționează clientul IPv6 către URL-ul corespunzător(Clientul în acest caz devenind serverul Linux pentru un alt server).
   - Verificare și gestionare a erorilor pentru comenzi neimplementate.
   - Comunicare HTTP IPv6 și transferul răspunsului către ClientIPv4_win_2023.exe, cu salvarea acestuia ca fișier .html în directorul propriu.

5. Scop:
   - Interoperabilitate între IPv4 și IPv6 pentru comunicare eficientă.
   - Utilizarea resurselor web IPv6 pentru comenzi specifice.
   - Gestionarea erorilor și furnizarea de mesaje informative către utilizator.
   - Înțelegerea mai bună în networking legat de IP alături de transmiterea datelor prin TCP și conexiunea realizată între 2 mașini la nivel de strat rețea și nu numai(Suita TCP/IP).

6. Lucruri aduse în plus separat cerințelor specifice:
   - Multithreading.
   - Manipularea întreruperei mascabile CTRL+C.
   - Datele transmise analizate prin WireShark – Suita TCP/IP analizată

Sarcina proiectului

Se da programul ClientIPv4_win_2023.exe care ruleaza pe statia fiecarei echipe sub sistemul de operare Windows 10 cu protocolul IPv4 activat. Se va realiza un program server/client in limbajul de programare C, sub sistemul de operare Linux, care va realiza urmatoarele functii:

a) Comunicare prin socket stream pentru IPv4 cu programul ClientIPv4_win_2023.exe. Programul server Linux se va realiza astfel incat sa astepte cereri de la clienti la portul 22GSE, unde G=numarul grupei, S=numarul semigrupei, E=numarul echipei. Clientul IPv4 se va putea conecta la toate adresele asignate serverului IPv4.

b) Preluare comanda "xy#" (xy=01, 02, …20) de la programul ClientIPv4_win_2023.exe. Clientul IPv6 se va conecta la URL (Uniform Resource Locator) al serverului IPv6 HTTP, ales in functie de "xy#" conform tabelului de mai jos:

xy# DESTINATIE COMANDA
01# www.yahoo.com
02# www.6init.org
03# www.ietf.org
04# www.pl.ipv6tf.org
05# www.a2hosting.com
06# www.traceroute6.net
07# www.speedtest6.com
08# www.ninux.org
09# ipv6-adresse.dk
10# www.openldap.org
11# www.pay4bugs.com
12# www.univie.ac.at
13# www.gentoo.org
14# he.net
15# www.afrinic.net
16# www.ipv6training.com
17# www.nanog.org
18# www.viagenie.ca
19# www.axu.tm
20# www.ja.net

c) In cazul in care comanda primita nu corespunde cu cerintele proiectului serverul Linux va retuna catre clientul IPv4 Windows un mesaj "Comanda neimplementata". Dupa stabilirea comunicarii prin socket stream IPv6, programul client IPv6 Linux va trimite catre programul Server IPv6 HTTP comanda GET / HTTP/1.0\r\n\r\n.

d) Preluarea raspunsului de la Server IPv6 HTTP, trimiterea acestuia catre programul ClientIPv4_win_2023.exe dar si salvarea lui in directorul propriu ca fisier cu extensia .html.

Observații

OBSERVATII+: Dupa realizarea conexiunii, programul Server IPv6 HTTP (denumire generica, in realitate el poate fi de exemplu un server apache) asteapta de la clientul IPv6 cererea pentru o pagina web, care va fi trimisa, dupa care inchide conexiunea.

1. Pentru compilare in Linux se va folosi compilatorul gcc.
2. Programul realizat va functiona ca server IPv4 pe partea de comunicatie cu programul ClientIPv4_win_2023.exe, iar pe partea de comunicatie cu Server IPv6 HTTP va functiona ca un client IPv6.
3. Se vor trata toate codurile de eroare returnate de functiile din program.
4. Se va folosi programul de captura Wireshark pentru a intelege ce adrese IPv4 + porturi TCP folosesc clientul si serverul IPv4.

Cerințe Redactarea Proiectului

• La sfarsitul fiecarui laborator se vor salva fisierele sursa si executabil in directorul propriu, dupa cum urmeaza: ~/etapa1 (dupa laborator 6), ~/etapa2 (dupa laborator 7), ~/etapa3 (dupa laborator8). In laboratorul 9 se va integra clientul IPv6 in serverul IPv4.

• Fiecare echipa va trebui sa cunoasca si sa explice rolul campurilor (head, body, a href) din pagina web adusa si salvata.

• Codul programului va trebui sa contina in mod OBLIGATORIU comentarii prin care acesta sa fie explicat.

• Fiecare echipa va TIPARI DOUA COPII continand codul sursa. In directorul propriu ~/final al masinii va fi salvata ultima versiune de program (inclusiv sursa). In Linux, cu ajutorul browserului lynx in mod text se va vizualiza pagina web salvata.

Cod

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <signal.h>

#define SERVER_IP_WEBSITE_ACCESS "2001:470:0:503::2"
#define MAX_SIZE_BUFF 1024
#define MAX_SIZE_BUFF_SPECIAL 100
#define PREDIFINED_PORT_22GSE 22311

// Indicator global pentru a indica închiderea serverului

volatile sig_atomic_t shutdown_flag = 0;

// Functie pentru a gestiona semnalul CTRL+C - Intrerupere Mascabila

void handle_ctrl_c(int sig) {
    printf("\nServerul se va închide în 15 secunde. Apăsați din nou CTRL+C pentru a forța
ieșirea\r\n");
    shutdown_flag = 1;
    /* Functia sleep() in C permite utilizatorilor sa astepte un thread curent pentru un
anumit timp.*/
    sleep(15);
    /*
    Dupa pauza de 15 secunde, se verifica valoarea variabilei shutdown_flag.
    Daca aceasta este incc setata la 1, se afiseaza un mesaj si programul se
    incheie cu succes (EXIT_SUCCESS).
    */
    if (shutdown_flag) {
        printf("\nInchidere fortata a Serverului\r\n");
        /*Functia void exit(int status) incheie procesul de apelare imediat.*/
        exit(EXIT_SUCCESS);
    }else
    {
        fprintf(stderr, "\nEROARE în gestionarea semnalului CTRL+C!!!Server
Neînchis!!!\r\n");
        exit(EXIT_FAILURE);
    }
}

/* Functia unde este gestionata comunicarea cu clientul.
  client_sock -  socket-ul client creat între statia Windows si serverul nostru Linux,
  sockaddr_in - structura de tip client pentru manipularea adreselor de internet */

void client(int client_sock, struct sockaddr_in *client_addr) {
    // Afiseaza un mesaj de initializare a programului
    printf("APELAREA FUNCTIEI CLIENT! INITIALIZARE!\r\n");

    // Buffer pentru a stoca și manipula secventa de caractere in memorie primita de la
procesul din
    // Windows pe un IP si un port specific - ENDPOINT
```

```
    char buffer[MAX_SIZE_BUFF];

    // Variabile pentru urmarirea bytes-ilor primiti și IP-ul clientului
    int bytes_received;
    char client_ip[INET_ADDRSTRLEN];
    int receive_function;

    // Conversia adresa IP a clientului intr-un format care poate fi usor citit - Din
binar în
    // format tip „ddd.ddd.ddd.ddd”

    inet_ntop(AF_INET, &(client_addr->sin_addr), client_ip, INET_ADDRSTRLEN);

    // Afisează informatii despre conexiune (IP client Windows si Port Windows - Acesta
din urma
    // asignat automat de sistemul de operare Windows)

    printf("Conexiune acceptată de la %s:%d\n", client_ip, ntohs(client_addr->sin_port));

    // In esenta, bucla ruleaza atat timp cat sunt primite cu
    // succes date de pe socket-ul asociat clientului
    while ((bytes_received = recv(client_sock, buffer, sizeof(buffer) - 1, 0)) > 0) {

        /*Sirurile de bytes terminate in nul marchează sfarsitul unui sir de caractere.*/

        buffer[bytes_received] = '\0';

        // Afiseaza comanda primita si informatii despre client-ul de IP: Port de unde a
fost primit

        printf("Comandă recepționată de la %s:%d: %s\n", client_ip, ntohs(client_addr-
>sin_port), buffer);

        // Se verifica daca comanda primita este „02#” si doar în acel moment se porneste
tot procesul
        // de proxy/intermediar efectuat de server, acum el devenind client pentru
        // alt server in urma trimiterii unui request the GET HTTP la care se primeste un
site web

        if (strcmp(buffer, "02#") == 0) {

            // Crearea unui nou socket pentru client - in cazul acesta, este
            // serverul Linux care devine client
            int instance_socket;

            /*
            socket() creeaza un endpoint (IP:PORT) pentru comunicare si returnează un
file descriptor
            care se refera la acel endpoint

            -> SOCK_STREAM
                Ofera secvential, fiabil, bidirectional, bazat pe conexiune
                fluxuri de bytes. Un mecanism de transmisie de date out-of-band
                poate fi sustinuta.
            */
            instance_socket = socket(AF_INET6, SOCK_STREAM, 0);
```

```
            // Se verifica daca crearea socketului a avut succes
            if (instance_socket == -1) {
                perror("EROARE LA CREARE SOCKET-ului CLIENTULUI!\r\n");
                // Program incheiat cu eroare - exit(EXIT_FAILURE)
                // Face flush la toate bufferele din program si inchide toate
                // programele asociate cu apelarea proiectului/procesului in cauza +
sterge
                // toate fisierele temporare
                exit(EXIT_FAILURE);
            } else
                printf("Client Socket Creat!\r\n");

            // Specifica adresa serverului pentru noul socket (IPv6) - In acest caz,
            // serverul alocat echipei noastre era 6init.org, dar din cauza faptului ca
            // acesta nu prezinta o adresă IPV6, ni s-a zis să folosim site-ul he.net,
acesta
            // avand adresa IPV6. Folositi/consultati programul
            //din fisierul ../final/tryhard pentru mai multe detalii

            /* Toate adresele IP ale site-ului www.he.net:
                IPv6: 2001:470:0:503::2
                IPv4: 216.218.236.2
            */

            /* Toate adresele IP ale site-ului www.6init.org:
                IPv4: 153.126.158.29
            */

            struct sockaddr_in6 server_address;
            server_address.sin6_family = AF_INET6;
            server_address.sin6_port = htons(80);
            inet_pton(AF_INET6, SERVER_IP_WEBSITE_ACCESS, &server_address.sin6_addr);

            /*
            Apelul de sistem connect() conecteaza socket-ul la care face referire
            file descriptor-ul sockfd (primul argument al functiei) la adresa specificata
de addr.
            */

            // Conectarea la server
            if (connect(instance_socket, (struct sockaddr*)&server_address,
sizeof(server_address)) == -1) {
                perror("EROAREA LA CONECTAREA SERVERULUI!\n\n");
                close(instance_socket);
                exit(EXIT_FAILURE);
            } else
                printf("Conectare Reușită la Serverul Căruia i Se Efectuează Requestul
HTTP\n\n");

            // Trimiterea request-ului HTTP GET catre server
            char *msg = "GET / HTTP/1.0\r\n\r\n";
            int len = strlen(msg);

            /*
            Apelul send() poate fi folosit numai atunci cand socket-ul este in
            stare conectata (astfel incat destinatarul vizat sa fie cunoscut).
            */
```

```
            // Se verifica daca operatiunea de trimitere a avut succes
            if (send(instance_socket, msg, len, 0) == -1) {
                perror("CLIENT: COMANDA NU A FOST TRIMISA! EROARE!\r\n");
            } else
                printf("CLIENT: COMANDA TRIMISĂ!\r\n");

            // Primirea datelor de la server si salvarea lor într-un fisier numit
„index.html”
            FILE *fp = fopen("index.html", "w+");
            if (!fp) {
                perror("EROARE ÎN DESCHIDEREA FIȘIERULUI!\r\n");
                close(instance_socket);
                exit(EXIT_FAILURE);
            }

            char buf[1];
            int bytes_received;

            // Scrierea caracter cu caracter in fisier cat timp date sunt primite
            while ((bytes_received = recv(instance_socket, &buf, 1, 0)) != 0) {
                if (bytes_received == -1) {
                    perror("EROARE ÎN RECEPȚIONAREA DATELOR!\r\n");
                    fclose(fp);
                    close(instance_socket);
                    exit(EXIT_FAILURE);
                }
                char temp = buf[0];
                fprintf(fp, "%c", temp);
            }

            // Inchiderea fisierul dupa primirea datelor
            if (fclose(fp) != 0) {
                perror("EROARE ÎN ÎNCHIDEREA FIȘIERULUI!\r\n");
                exit(EXIT_FAILURE);
            }

            // Inchiderea socket-ului intre serverul/client (Server HTTP, Client Linux)
            // pentru a restrictiona operatiunile ulterioare de trimitere/primire
            if (shutdown(instance_socket, 2) == 0)
                printf("\n");

            // Manipulare speciala in alt buffer - Necesar pentru afisarea codului HTML
            // in procesul/thread-ul de pe client-ul Windows
            char special_buffer[MAX_SIZE_BUFF_SPECIAL];
            char copie_buffer[MAX_SIZE_BUFF_SPECIAL];
            receive_function = recv(client_sock, special_buffer, sizeof(special_buffer),
0);

            // Deschidem fisierul „index.html” pentru citire
            FILE *fd = fopen("index.html", "r");
            if (!fd) {
                perror("EROARE ÎN CITIREA FIȘIERULUI!");
                exit(EXIT_FAILURE);
            }

            char data[50];
```

```
            int cititorul, comparatie;

            // Verificam daca în buffer e indicat deconectarea clientului cumva
            comparatie = strncmp(special_buffer, "DISCONNECT", receive_function);
            if (comparatie == 0) {
                printf("Din păcate, CLIENT ABRUPT DECONECTAT!!!\n\n");
                break;
            } else {
                // Verifica daca buffer-ul special indică o comanda specifică - 02# în
acest caz
                int noua_comparatie = strncmp(special_buffer, "02#", receive_function);
                if (noua_comparatie == 0) {
                    // Trimitem continutul „index.html” catre client în blocuri de 50 de
caractere
                    while ((cititorul = fread(data, 1, 50, fd)) > 0) {
                        send(client_sock, data, cititorul, 0);
                    }
                    fclose(fd);

                    // Modifica si trimite un raspuns catre client pe baza buffer-ului
special primit
                    // unde se indică faptul ca, comanda este implementată
                    size_t buffer_len = strlen(special_buffer);
                    if (buffer_len >= 3) {
                        strncpy(copie_buffer, special_buffer, 3);
                        copie_buffer[3] = '\0';
                        strcat(copie_buffer, " este implementata \n");
                        printf("Comanda Implementa \n");
                        send(client_sock, copie_buffer, strlen(copie_buffer), 0);
                    }

                    // Realizam un clear la buffer-ul special folosit
                    special_buffer[0] = '\0';
                } else {
                    // Modifica si trimite un raspuns catre client pe baza buffer-ului
special primit
                    // unde se indica faptul ca, comanda nu este implementata
                    size_t buffer_len = strlen(special_buffer);
                    if (buffer_len >= 3) {
                        strncpy(copie_buffer, special_buffer, 3);
                        copie_buffer[3] = '\0';
                        strcat(copie_buffer, " nu este implementata \n");
                        printf("Comanda nu este implementata \n");
                        send(client_sock, copie_buffer, strlen(copie_buffer), 0);
                    }

                    // Realizam un clear la buffer-ul special folosit
                    special_buffer[0] = '\0';
                }
            }
            printf("Request HTTP GET Trimis și procesul a fost până la capăt executat cu
succes!\n");
        } else {
            // Raspuns implicit pentru alte comenzi in afara de cel asignat echipei
            printf("Comanda Implementata: %s\n", buffer);
            // Trimite un raspuns clientului aferent altor comenzi in afara de cel
asignat
```

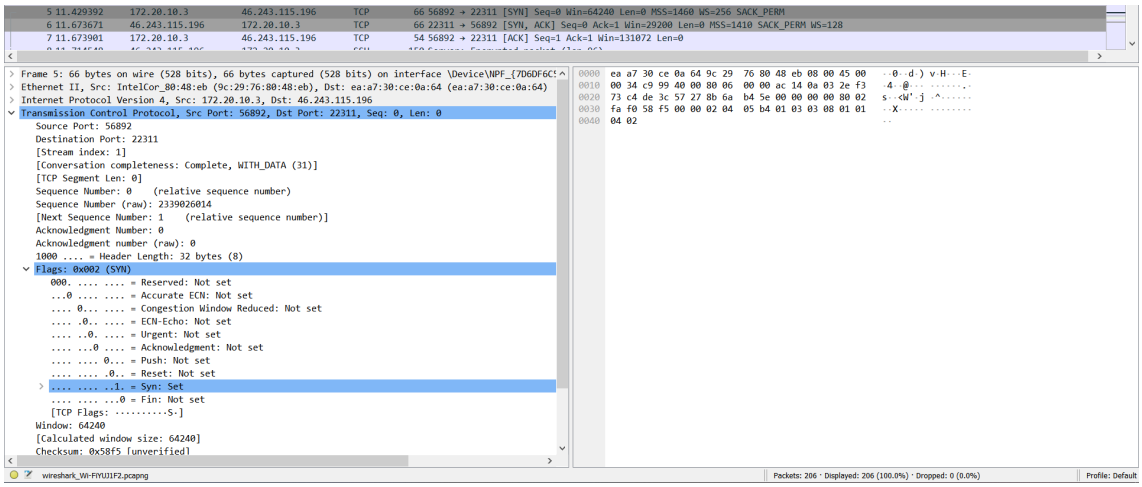
```
            send(client_sock, "Comanda Nerecunoscuta", strlen("Comanda Nerecunoscuta"),
0);
        }
    }
}

// Funcția Main
int main() {
    /*
    signal() - Stabileste modul in care este gestionat semnalul.
    Poate fi configurat sa gestioneze implicit semnalul, sa-l ignore
    sau sa apeleze o functie specificata de utilizator, cum ar fi "handle_ctrl_c(int
sig)" in acest caz.
    */

    // Configurarea gestionarii semnalului pentru CTRL+C - Necesar sa fie executat
initial
    signal(SIGINT, handle_ctrl_c);

    printf("Program Inițializat! Nuoah, să îi dăm drumu'\r\n");

    // Variabile pentru a stoca socket-urile de client si server - in acest caz, clientul
este Host-ul
    // cu Windows, iar serverul este Host-ul Linux
    int server_sock, client_sock;

    // Crearea unui socket pentru server (IPv4)
    server_sock = socket(AF_INET, SOCK_STREAM, 0);

    // Verificam daca crearea socket-ului serverului a avut loc cu succes
    if (server_sock < 0) {
        perror("EROARE LA CREAREA SOCKET-ului SERVERULUI!\r\n");
        exit(EXIT_FAILURE);
    } else
        printf("Server Socket Creat!\n");

    // Specifica adresa si portul serverului (IPv4)
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_len = sizeof(client_addr);

    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PREDIFINED_PORT_22GSE);
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);

    int yes = 1;

    /*
    setsockopt() - Necesitatea lui în program ==>
    Un alt lucru la care trebuie sa fim atenti când apelati bind():

    Uneori, se observa faptul ca se încerca repornirea server-ului și bind() esueaza,
    pretinzand „Adresa deja utilizata”. Socket-ul care a fost conectata inca mai "atarna"
    in kernel si ocupa portul. Se putea rezolva fie prin asteptare pana cand se sterge,
    fie sa adaugam cod la program, permitându-i sa refoloseasca portul
    */
```

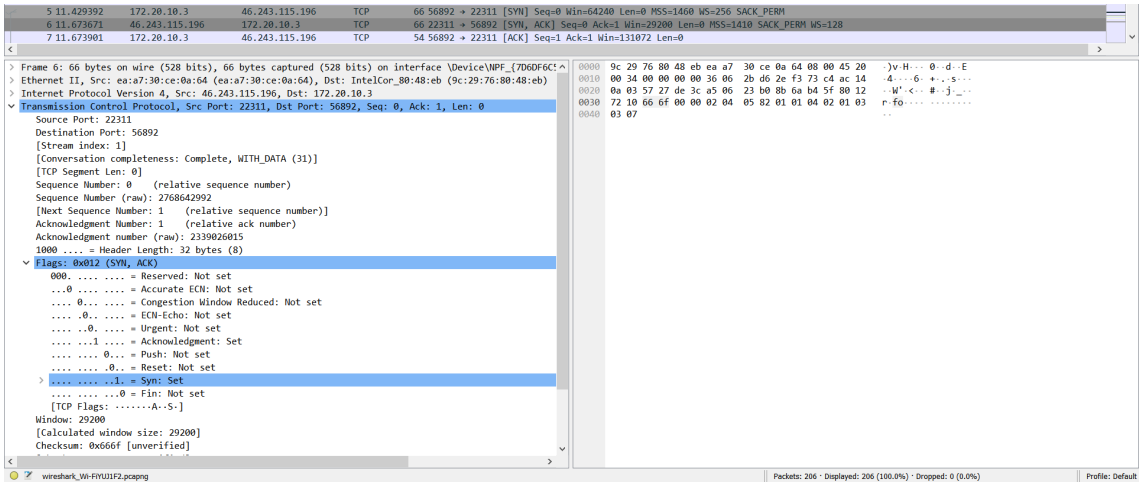
```
    // Seteaza opțiunile de socket pentru a evita eroarea „Adresa deja in uz”.
    if (setsockopt(server_sock, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof yes) == -1) {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }

    int n;

    // Realizam bind pe socket-ul serverului pe adresa si portul specificat
    n = bind(server_sock, (struct sockaddr*) &server_addr, sizeof(server_addr));

    // Verificam daca bind a fost cu succes
    if (n < 0) {
        perror("EROARE DE ASOCIEREA SOCKET-ului!\r\n");
        exit(EXIT_FAILURE);
    } else
        printf("Asociere efectuată! Socket pe Portul %d\n", PREDIFINED_PORT_22GSE);

    // Ascultam conexiunile pentru cereri de la client
    int listen_var = listen(server_sock, 10);

    // Verificam daca listen() se realizeaza cu succes
    if (listen_var == -1) {
        perror("EROARE ÎN CADRUL LISTENING!\r\n");
        shutdown(server_sock, 2);
        exit(EXIT_FAILURE);
    } else
        printf("Listen cu succes. Serverul Ascultă pe Portul %d\n",
PREDIFINED_PORT_22GSE);

    // Bucla infinita pentru a accepta continuu conexiunile client de intrare
    while (1) {

        // Acceptam o conexiune de client
        client_sock = accept(server_sock, (struct sockaddr *)&client_addr, &client_len);

        // Verificam daca clientul a fost conectat cu succes
        if (client_sock == -1) {
            perror("EROARE ACCEPTÂND CONEXIUNI!\r\n");
            exit(EXIT_FAILURE);
        } else {

            /*
            fork() - The multithreading status quo

            fork() creeaza un nou proces duplicând procesul apelant.
            Noul proces este numit proces copil(child), iar procesul
            apelant este numit proces parinte(parent).

            Procesul copil si procesul parinte ruleaza in spatii de memorie separate.
            La momentul fork(), ambele spatii de memorie au continut identic.

            fork() creeaza un nou proces care este o copie exacta a procesului de
apelare.
            Noul proces este procesul copil; vechiul proces este procesul parinte.
            Copilul primește un ID de proces nou, unic. fork() returneaza un 0 procesului
            copil si ID-ul procesului copilului parintelui. Valoarea returnata este modul
```

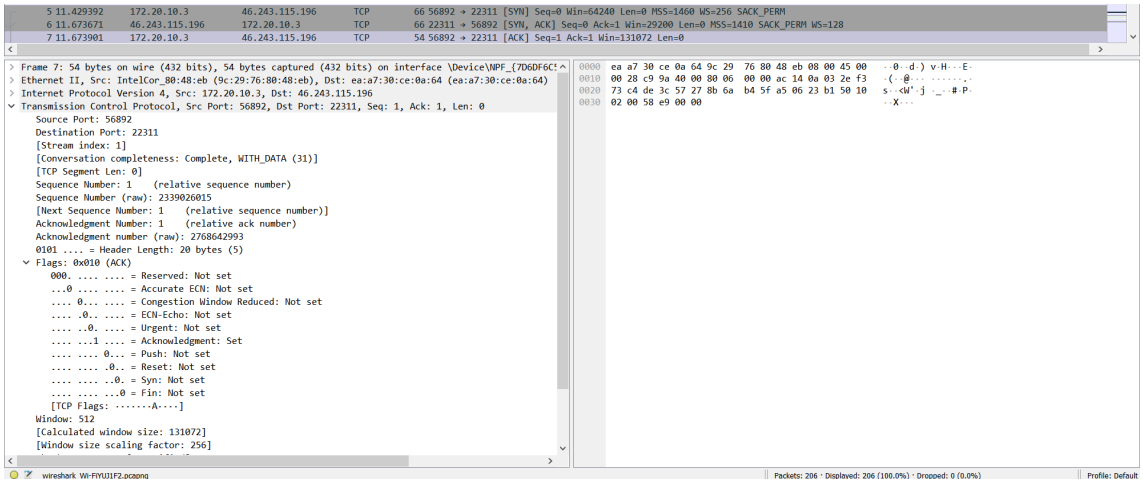
```
            in care un program bifurcat determina dacă este procesul parinte sau procesul
copil.

            */

            // Gestionarea clientul intr-un proces separat pentru a permite procesarea
concomitenta
            // a multiplilor clienti aparuti
            pid_t pid = fork();

            // Verificam daca procesul fork a fost realizat cu succes
            if (pid == -1) {
                perror("EROARE ÎN PROCESUL DE MULTITHREADING!!!\r\n");
                close(client_sock);
            } else if (pid == 0) {
                // Procesul Child servește clientul
                client(client_sock, &client_addr);
                close(server_sock); // Închidem procesul child
            } else {
                // Procesul parinte continua sa accepte conexiuni
                close(client_sock); // inchidem procesul parinte
            }
        }
    }

    // Inchidem socket-ul serverului
    if (shutdown(server_sock, 2) == 0)
        printf("Shutdown Server Socket\n");

    return 0;

    // END OF PROGRAM
}
```

Capturi WireShark:

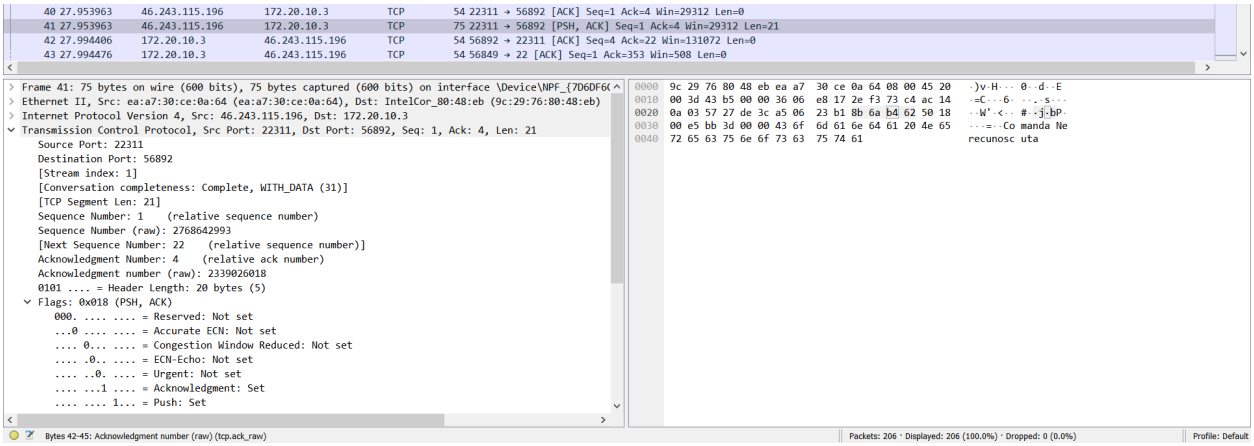
1. Realizarea Conexiunii TCP: SYN -> SYN+ACK -> ACK



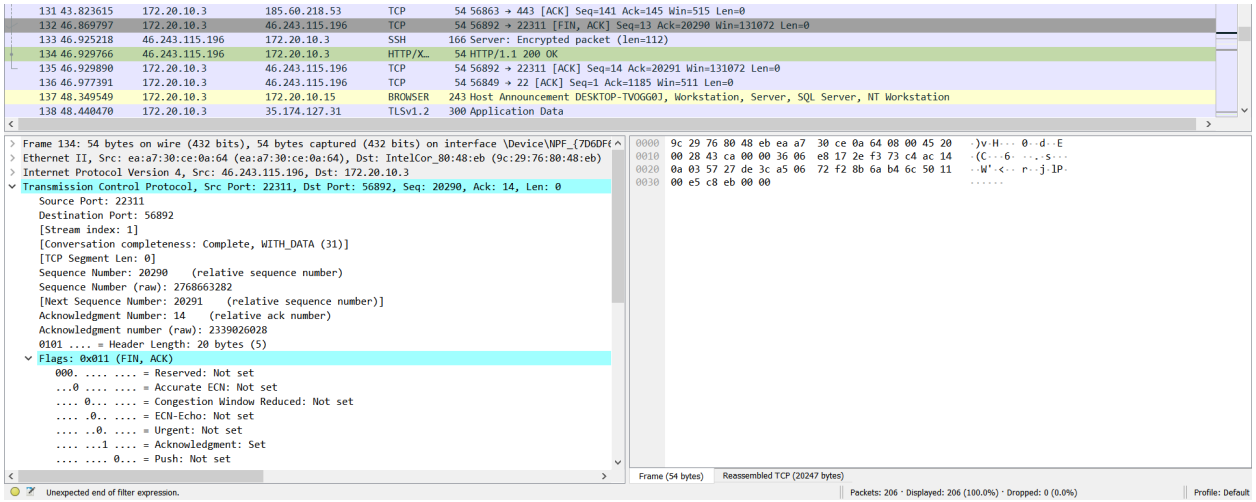




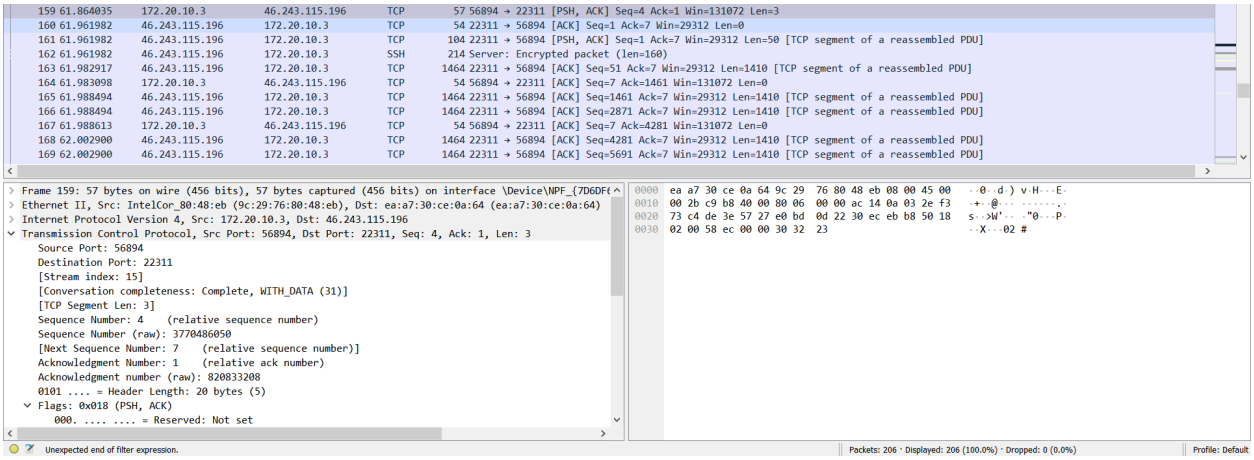
2 . Trimiterea unei comenzi neimplementate pentru verificare:

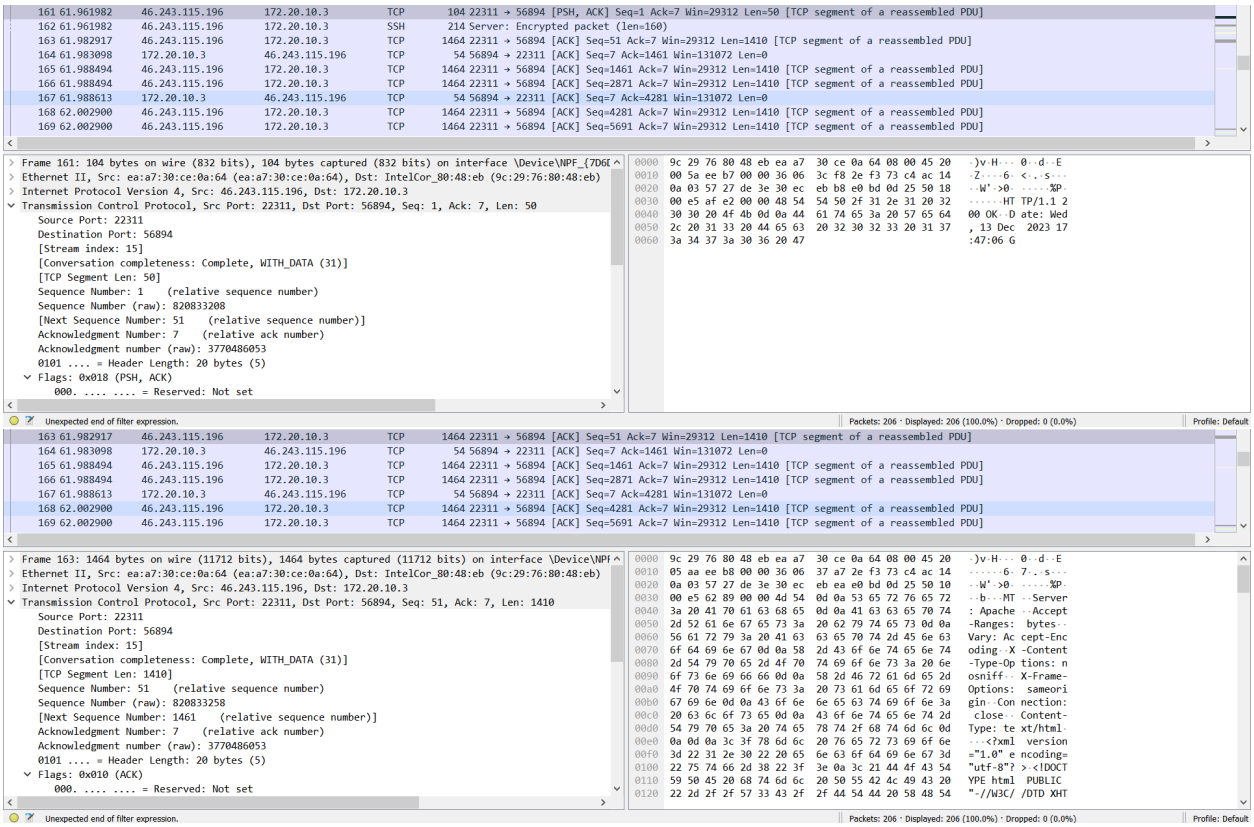


3 . Trimiterea request-ului HTTP GET către Serverul Apache IPv6 – www.he.net



4 . Prin trimiterea comenzii implementate din nou, adică „#02”, se va trimite client-ului Windows codul HTML al site-ului pe endpoint-ul său (172.20.10.3:56892)





5 . Deconectarea din partea clientului. Închiderea conexiunii TCP tip full-duplex