Simple Web Server

Radu Emilian

Universitatea Alexandru Ioan Cuza Iasi, Facultatea de Informatica
mihai.radu@info.uaic.ro

Abstract. Un simplu server web realizat prin socketi.

Keywords: TCP · HTTP

1 Introducere

1.1 Cerinta Proiectului

Realizati un server concurent ce trimite fisiere (.txt sau .html fara scripturi) din directorul curent catre orice browser se conecteaza la el. Server-ul va trebui sa trimita antete HTTP corecte. Nu implementati un client: server-ul va trebui sa interactioneze corect cu orice browser.

1.2 Modul de abordare

Pentru rezolvarea cerintei se va utiliza drept punct de "start" fisierul servTcpIp.c de pe site-ul cursului, la care se vor face modificari pentru a accepta conetarea unui client de tipul "browser".
Primul pas va fi transformarea serverului initial ce este de tip iterativ intr-un server concurent, pentru a accepta conexiunea mai multor clienti consecutiv.
De asemenea, se va crea fisierul "config.txt" ce contine detalii esentiale pentru functionarea serverului.
Fisierele ce vor fi furnizate clientului de catre server vor avea locatia definita in fisierul de configurare specificat anterior si vor avea extensia .txt, .html, .jpg sau .png (configurabile in fisierul de configurare).

1.3 Motivatia alegerii proiectului

2 Tehnologii Utilizate

1. TCP/IP Concurent
2. HTTP Protocol

2.1 TCP/IP Concurent

A fost utilizat protocolul Transmission Control Protocol (TCP) deoarece alternativa (UDP) ar fi cauzat pierderi de informatii esentiale in cadrul transferului de fisiere catre clienti. Informatia ceruta trebuie sa ajunga in mod exact la clientul ce o cere iar acest aspect este facilitat doar de protocolul TCP.

2.2 HTTP Protocol

Singura modalitate de a transmite fisiere catre un browser este prin intermediului protocolului HTTP sau Hypertext Transfer Protocol ce este protocolul utilizat de World Wide Web.

3 Arhitectura aplicatiei

3.1 Concepte implicate

In cadrul implementarii am modularizat codul, impartindu-l in functii pentru a fi usoara atat citirea cat si intelegerea programului. Codul este impartit in 4 functii :

1. void server_setup(struct configFile *config);
2. void get_client(int client, struct configFile *config);
3. void send_request(int client, struct htmlrequest *request, struct configFile *config);
4. int main(int argc, char **argv)

3.2 Diagrama Aplicatiei

4 Detalii de implementare

4.1 Cod relevant

Pentru a retine toate informatiile despre server se va folosi urmatoarea structura:

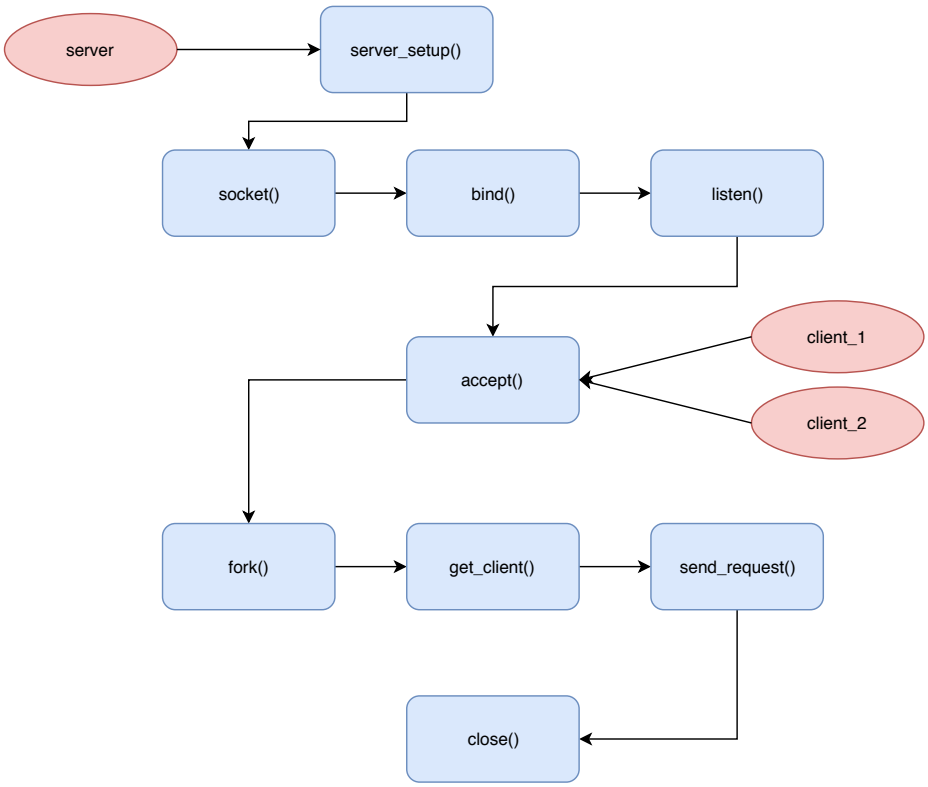
```
struct configFile
{
    int port;
    char address[PATH_DOC];
    char request[MAXDAT][MAXDAT];
    char send[MAXDAT][MAXDAT];
    char default_page[MAXDAT];
};
```

Cererea clientului reprezentat de browser va fi salvata in structura:

```
struct htmlrequest
{
    char *full, *url, *version, *method;
};
```

Acest aspect se va realiza pentru fiecare client in parte, asa ca la finalul cererii se va goli structura prin:

```
free(request.full);
free(request.url);
free(request.method);
free(request.version);
```


server
server_setup()
socket()
bind()
listen()
client_1
accept()
client_2
fork()
get_client()
send_request()
close()

Se va folosi citirea din fisiere destul de frecent:

```
    FILE *file_config;
    file_config = fopen(path_file, "r");

    if (file_config == NULL)
    {
        perror("[SERVER] Eroare la deschiderea fisierului de
    configurare: ");
        exit(ERROR);
    }
‘
    if (file_config)
    {
        fseek(file_config, 0, SEEK_END);
        length = ftell(file_config);
        buff = malloc(length);
        fseek(file_config, 0, SEEK_SET);
        if (buff)
            fread(buff, 1, length, file_config);
        fclose(file_config);
    }
```

Configuratia serverului se va citi in functia server_config:

```
void server_setup(struct configFile *config)
```

Dupa ce s-au citit detaliile din fisierul config.txt serverul va realiza bind(), socket() si listen():

```
    if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == ERROR)
    {
        perror("[SERVER] Eroare la socket().\n");
        return errno;
    }

    if (bind(sd, (struct sockaddr *)&server, sizeof(struct
    sockaddr)) == ERROR)
    {
        perror("[SERVER] Eroare la bind().\n");
        return errno;
    }

    if (listen(sd, MAXCL) == ERROR)
    {
        perror("[SERVER] Eroare la listen().\n");
        return errno;
    }
```

Clientii vor fi serviti intr-o bucla while() in care se realizeaza fork() pentru a-i servi concurent:

```
  if ((pid = fork()) == ERROR)
```

```
        {
            perror("[SERVER] Eroare la fork(). \n");
        }
```

Functia ce se ocupa de rezolvarea cerintelor clientului este

```
void get_client(int client, struct configFile *config)
```

Urmatorul pas in servirea clientului dupa ce s-a aflat cererea este trimiterea informatiilor din fisierul cerut:

```
void send_request(int client, struct htmlrequest *request,
    struct configFile *config)
```

Rezolvarea cererii se face prin intermediul protocolului HTTP 1.1 (cea mai recenta versiune) cu ajutorul string-urilor:

```
    char html[] = "HTTP/1.1 200 OK:\r\n"
            "Content-Type: text/html; charset=UTF-8\r\n\r\n";
```

La final, fisierul cerut este transmis prin send()

```
send(client, buff, read_bytes, 0);
```

4.2 Scenarii de utilizare

1. Un client doreste sa acceseze pagina de index a serverului.
2. Un client doreste sa acceseze un fisier specific cu extensia .txt/.html.
3. Un client doreste sa acceseze o imagine cu extensia .jpg/.png
4. Un client doreste sa acceseze un fisier care nu exista in root.

5 Concluzii

Conform implementarii actuale a serverului, din fisierul de configurare nu se preiau vectorii request si send, astfel extensiile fisierelor fiind hard-coded. Pe viitor doresc rezolvarea acestei probleme pentru a putea fi adaugat in fisierul de configurare oricate extensii - inclusiv .css sau .js.
Deoarece mi-am ales acest proiect am reusit sa invat mai multe aspecte ale limbajului C pe langa concepte necesare obiectului Retele de Calculatoare, ce ma vor ajuta atat in crearea/intretinerea unui server pe viitor.

References

1. Computer Networks - Faculty of Computer Science, `https://profs.info.uaic.ro/~computernetworks/`
2. servTCPIt.c, Lenuta Alboaie, adria@infoiasi.ro, (c)2009
3. Hypertext Transfer Protocol – HTTP/1.1, `https://www.w3.org/Protocols/rfc2616/rfc2616.html`
4. A template to make good README.md, `https://gist.github.com/PurpleBooth/109311bb0361f32d87a2`

5. C Library - ¡string.h¿, https://www.tutorialspoint.com/c_standard_library/string_h.htm
6. Information for Authors of Springer Computer Science Proceedings, https://www.springer.com/gp/computer-science/lncs/conference-proceedings-guidelines
7. How to read the content of a file to a string in C?, https://stackoverflow.com/questions/174531/how-to-read-the-content-of-a-file-to-a-string-in-c
8. Transmission Control Protocol, https://en.wikipedia.org/wiki/Transmission_Control_Protocol
9. Berkeley sockets, https://en.wikipedia.org/wiki/Berkeley_sockets