EA872 Laboratório de Programação de Software Básico
Atividade 8

Vinícius Esperança Mantovani **RA 247395**

Entrega (limite): 04/10/2023, 13:00.

Código fonte desenvolvido:

Neste laboratório, desenvolvi um código de um servidor web capaz de responder a requisições feitas por meio de um browser. O código desenvolvido é o seguinte:

```
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <string.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/stat.h>

/**
* @brief Trata a mensagem recebida em buf1, passa-a para o buf2 depois de tratada e imprime o conteudo de buf2 no soquete
*
* @param buf1 string contenedora da mensagem a ser tratada
* @param buf2 string que recebe a mensagem tratada
* @param soquete socket em que se escreve a mensagem
*/
int processar_mensagem(char *buf1, char *buf2, int soquete){
   char caminho[100]; //usado para armazenar o caminho do recurso a ser passado pelo soquete
```

```
   int teste_while; //usado para testar se o while ja deve ser parado
   struct stat statbuf, statind; //usadas para fazer condicionais com
dados de arquivos
   strcat(buf1, "\0");
   strcat(buf2, "\0");
   strcpy(caminho, "\0");
   char *token = strstr(buf1, "/"); //procura a primeira ocorrencia de
"/" na mensagem de entrada e salva o ponteiro dessa ocorrenccia em
token

   if(token != NULL){ //caso haja ocorrencia de "/"

       char *token_complete = strtok(token, " "); //procura pela
primeira ocorrencia de expaco na string token e limita essa string ate
esse ponto, passando o novo apontador para token_completo

       strcpy(caminho, "/home/pininsu/meu-webspace\0"); //caminho
inicialmente eh o caminho para o webspace em minha maquina
       if(strcmp(token_complete, "/") !=0){ //caso token_complete nao
seja igual a "/"
           strcat(caminho, token_complete); //concatena token_complete
em "caminho" para completar o caminho ate o recurso requerido
       }

       int st_out = stat(caminho, &statbuf); //adquire dados a respeito
do arquivo requerido

       if (st_out == -1 && errno == ENOENT) { //caso nao exista o
arquivo
       //imprime a saida necessaria (resposta ao cliente)
           strcpy(caminho,
"/home/pininsu/meu-webspace/erro404.html");//altera a string caminho
para conter o caminho para o html de erro 404
           write(soquete, "HTTP/1.1 404 File Not Found\r\nDate: Thu Sep
14 17:54:08 2023\r\nServer: Servidor HTTP ver. 0.1 de Vinicius
Esperanca Mantovani\r\nConnection: keep-alive\r\nLast-Modified: Wed Sep
6 22:24:38 2023\r\nContent-Length: 283\r\nContent-Type:
text/html\r\n\r\n", 239);//printa o cabecalho de resposta no soquete
       } else if ((statbuf.st_mode & S_IRUSR) == 0){  //caso nao haja
permissao para leitura do arquivo
           strcpy(caminho,
"/home/pininsu/meu-webspace/erro403.html");//altera a string caminho
para conter o caminho para o html de erro 403
```

```
            write(soquete, "HTTP/1.1 403 Forbidden\r\nDate: Thu Sep 14
17:54:08 2023\r\nServer: Servidor HTTP ver. 0.1 de Vinicius Esperanca
Mantovani\r\nConnection: keep-alive\r\nLast-Modified: Wed Sep  6
22:24:38 2023\r\nContent-Length: 283\r\nContent-Type:
text/html\r\n\r\n", 234);//printa o cabecalho da resposta no soquete
        } else { //caso o arquivo exista e seja legivel
            write(soquete, "HTTP/1.1 200 OK\r\nDate: Thu Sep 14 17:54:08
2023\r\nServer: Servidor HTTP ver. 0.1 de Vinicius Esperanca
Mantovani\r\nConnection: keep-alive\r\nLast-Modified: Wed Sep  6
22:24:38 2023\r\nContent-Length: 283\r\nContent-Type:
text/html\r\n\r\n", 227);//printa o cabecalho da resposta no soquete

            if((statbuf.st_mode & S_IFMT) == S_IFDIR){ //caso seja um
diretorio
                char cpy1[100]; //usado para conter o caminho ate
index.html
                strcpy(cpy1, caminho);
                strcat(cpy1, "/index.html");
                int st_cpy = stat(cpy1, &statind); // st_cpy usado para
condicionais de existencia e statind para condicional de permissao para
leitura
                if(st_cpy != -1 && (statind.st_mode & S_IRUSR) != 0){
//caso index.html exista e tenha permissao para leitura
                    strcpy(caminho, cpy1);
                } else if(st_cpy == -1 && errno == ENOENT){ //caso
index.html nao exista
                    strcpy(caminho,
"/home/pininsu/meu-webspace/erro404.html");
                    write(soquete, "HTTP/1.1 404 File Not Found\r\nDate:
Thu Sep 14 17:54:08 2023\r\nServer: Servidor HTTP ver. 0.1 de Vinicius
Esperanca Mantovani\r\nConnection: keep-alive\r\nLast-Modified: Wed Sep
6 22:24:38 2023\r\nContent-Length: 283\r\nContent-Type:
text/html\r\n\r\n", 239);
                } else if(st_cpy != -1 && (statind.st_mode & S_IRUSR) ==
0){ //caso index.html exista mas nao tenha permissao para leitura
                    strcpy(caminho,
"/home/pininsu/meu-webspace/erro403.html");
                    write(soquete, "HTTP/1.1 403 Forbidden\r\nDate: Thu
Sep 14 17:54:08 2023\r\nServer: Servidor HTTP ver. 0.1 de Vinicius
Esperanca Mantovani\r\nConnection: keep-alive\r\nLast-Modified: Wed Sep
6 22:24:38 2023\r\nContent-Length: 283\r\nContent-Type:
text/html\r\n\r\n", 234);
                }
```

```
            }
        }
        //write(1, caminho, strlen(caminho)); //teste para ver se o
caminho chega certo ate aqui

        int fd = open(caminho, O_RDONLY, 0600); //file descriptor usado
para imprimirmos o conteudo do arquivo

        do{
            fcntl(fd, F_SETFL, O_NONBLOCK); //define fd como nao
bloqueante para ajudar na condicao de parada do while
            teste_while =  read(fd, buf2, 1023); //le o conteudo do
arquivo para o buffer2
            write(soquete, buf2, teste_while); //escreve o conteudo do
buffer2 no soquete
        } while(teste_while == 0); //se nao ha mais o que ser lido do
arquivo, para o while
    }
}

int main()
{
    int teste_while; //usado para teste de parada do while
    int soquete, soquete_msg, tamanhobuf=1024;
    struct sockaddr_in meu_serv, meu_cli;
    int tam_endereco = sizeof(meu_cli);
    char buf_entrada[tamanhobuf], buf_saida[5000];
    soquete = socket(AF_INET, SOCK_STREAM, 0); // 0 indica "Use
protocolo padrão"
    meu_serv.sin_family = AF_INET;
    meu_serv.sin_port = htons(3030); // htons() converte valores para
representação de rede
    meu_serv.sin_addr.s_addr = INADDR_ANY; // qualquer endereço válido
    bind(soquete, (struct sockaddr*)&meu_serv, sizeof(meu_serv));
    listen(soquete, 5); // prepara socket e uma lista para receber até 5
conexões

    while(1)
    {
        soquete_msg = accept(soquete, (struct sockaddr*)&meu_cli,
&tam_endereco);

        do
```

```
        {
            fcntl(soquete_msg, F_SETFL, O_NONBLOCK); //seta o soquete
como nao bloqueante para ajudar na condicao de parada do while
            teste_while = read (soquete_msg, buf_entrada, tamanhobuf);
//le o soquete e passa a informacao para o buffer de entrada
        }   while(teste_while == 0); //para o while caso nao haja mais o
que ser lido

        processar_mensagem(buf_entrada, buf_saida, soquete_msg); //entra
na funcao de processamento de requisicao e producao de resposta

        close(soquete_msg);
    }
}
```

Vale então, dar uma descrição passo a passo do código. De começo, temos a definição de várias variáveis que são usadas para permitir o funcionamento do programa. Entre elas, estão algumas que não são parte do processo de criação do servidor, mas sim parte da criação de respostas às requisições passadas por um cliente, são elas: buf_entrada e buf_saida. A primeira recebe, ainda na main, o conteúdo passado pelo soquete_msg, soquete responsável por receber as requisições do cliente. Em seguida, são passados ambos os buffers e, também o soquete_msg como argumentos para a função processar_mensagem.

Dentro da função processar_mensagem, temos, inicialmente, a definição de uma string que contém todo o conteúdo da string buf_entrada a partir da primeira ocorrência de “/”, por meio da função strstr. Em seguida, verifica-se se a string retornada pela função não é nula, ou seja, se existe alguma ocorrência de “/” em buf1 e, em caso afirmativo a string é limitada até antes da primeira ocorrência de “ ”, por meio da função strtok, essa string é “token_complete”. Agora, inicia-se o processo de produção da mensagem de resposta do servidor ao cliente. Isto se dá de modo que, primeiramente é copiado para uma string (caminho) o caminho até o webspace presente em minha máquina e, em seguida, no caso de a string token_complete ser diferente de “/”, então ela é concatenada à string caminho.

Seguindo, temos a verificação a respeito de o arquivo ser um arquivo diretório e, em caso afirmativo: Se ele contém o arquivo index.html, então o nome “index.html” é concatenado à string caminho; Se ele contém o arquivo index.html, mas este arquivo não tem permissão para leitura, então é impressa uma mensagem de erro 403 no soquete e o nome do arquivo de erro 403 é concatenado à string caminho; Se ele não contém o arquivo, então é impressa uma mensagem de erro 404 e é concatenado o nome do arquivo de erro 404 à string caminho.

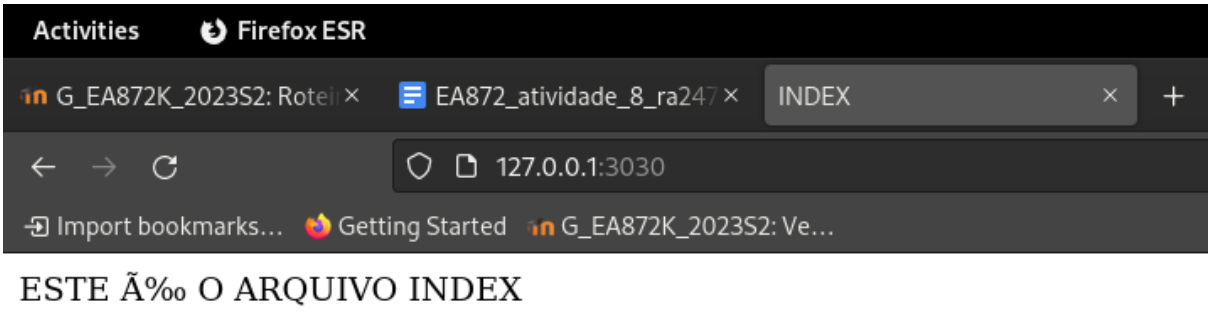
Em seguida, é impresso o arquivo de acordo com o que foi posto na string “caminho” anteriormente, resultando naquilo que se vê mais adiante no relatório.

Instruções de compilação e de uso do código:

Para que tenha o funcionamento adequado em uma máquina que não seja a minha, é necessário alterar a ocorrência de “/home/pininsu/meu-webspace” no código e compilá-lo usando gcc, conforme se segue: “gcc -o servidor servidor.c”. Para executá-lo, então, basta usar: “./servidor”.

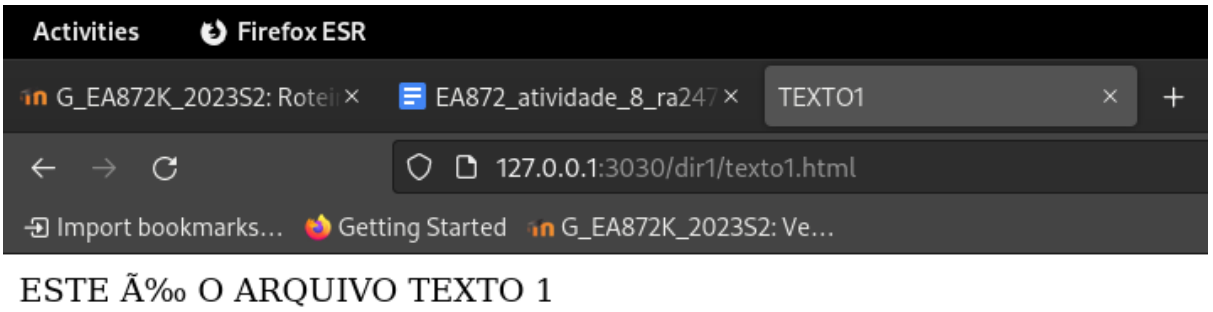
Testes indicando sucesso:

Inicialmente, temos um teste para uma requisição da página padrão pelo browser:



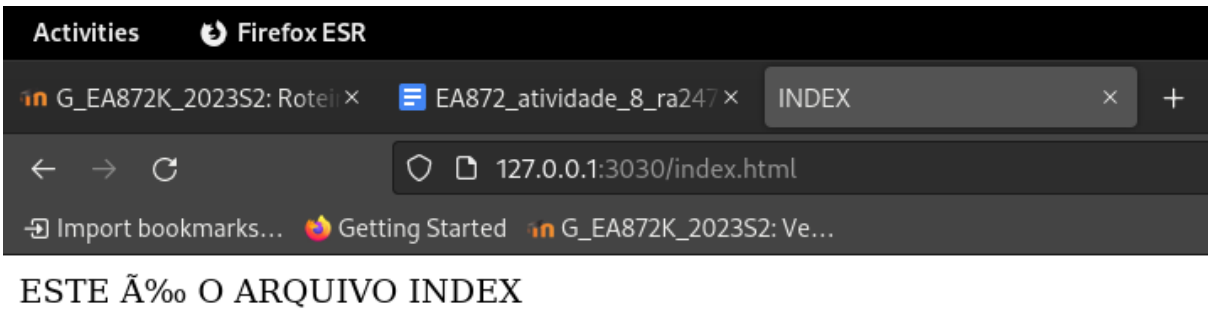
Este é o retorno obtido e, é o resultado de index.html.

Agora, temos o teste para a busca pelo arquivo “/dir1/texto1”:



Este é o retorno obtido e, é o resultado de texto1.html.

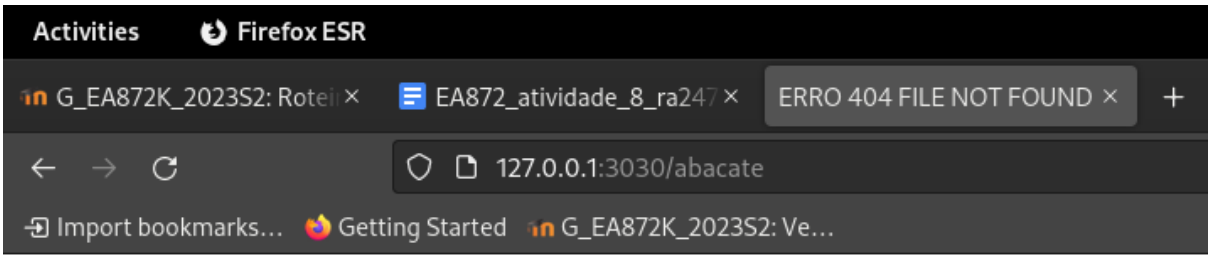
Por fim, um terceiro exemplo de teste é para o arquivo index.html buscado explicitamente:



Mesmo resultado do primeiro teste mas para pesquisa diferente.

Exemplo de teste para erro de recurso não existente:

Ao tentar buscar por “abacate.html”, temos a seguinte resposta do servidor:



Exemplo de teste de acesso negado:

Ao tentar acessar “/dir1/texto2.html”, temos o seguinte: