INSTITUTO FEDERAL
Paraná
Campus Pinhais

Informática
Redes de computadores — 28 de abril de 2025
Atividade: servidor HTTP — v1.0

1 Cliente e servidor HTTP

Seu trabalho é implementar um cliente e um servidor HTTP, ambos em formatos simplificados. Seu cliente deve fazer requisições de arquivos ao servidor e mostrar a resposta na tela.

O servidor deve aceitar apenas as requisições obrigatórias (GET e HEAD) e entregar o arquivo pedido ao cliente. O servidor deve detectar e informar ao cliente algumas situações de erro, descritas abaixo.

2 Detalhamento do cliente

A partir do exemplo mostrado em sala e disponível na página (`tcp_simple_client.c`), você deve criar o seu cliente HTTP em C.

O seu cliente deve receber 4 argumentos no momento da execução: o servidor e porta a se conectar, o tipo da requisição HTTP e o objeto a ser requisitado. O cliente deve ser executado de forma parecida a: `./client localhost 8080 GET /index.html`

O seu cliente deve mandar uma requisição no formato

```
GET /index.html HTTP/1.1
Host: localhost
```

Importante: as quebras de linha devem ser `\r\n` e deve haver uma linha em branco no final!

2.1 Passos sugeridos para construção do cliente

1. Copiar o `tcp_simple_client.c`
2. Alterar o valor de `BUFFER_SIZE` para `BUFSIZ`
3. Ajustar a quantidade de argumentos conforme descrito acima
4. Criar a requisição
   (a) Crie uma string para a requisição: `char request[BUFFER_SIZE];`
   (b) Use a função `snprintf` para criar a requisição a partir de outras variáveis. Essa função retorna o tamanho da string construída: você vai precisar!
5. Envie a requisição para o servidor (`send`)
6. Receba a resposta do servidor e imprima na tela
   - Aqui temos um problema pois o servidor pode mandar a resposta dividida em várias etapas
   - Por isso, você deve tentar receber informações do servidor usando `recv` até a função retornar 0 (não há mais o que receber)
   - A ideia geral desta parte é:

```
bytes_recebidos = recv();
while (bytes_recebidos > 0) {
    printf();
    bytes_recebidos = recv();
}
```

3 Detalhamento do servidor

A partir do exemplo mostrado em sala e disponível na página (`tcp_simple_server.c`), você deve criar o seu servidor HTTP em C.

O seu servidor deve receber 1 argumento no momento da execução: a porta em que aceitará conexões. O servidor deve ser executado de forma parecida a: `./server 8080`

Seu servidor deve aceitar apenas dois tipos de requisições HTTP:

- GET: envia o arquivo requisitado
- HEAD: envia informações do arquivo requisitado, mas não envia o arquivo

3.1 Passos sugeridos para construção do servidor

1. Copiar o `tcp_simple_server.c`
2. Alterar o valor de `BUFFER_SIZE` para `BUFSIZ`
3. Modificar o `while` da função principal para que fique no formato:

```
while (true) {
    client_connection_socket = accept(server_socket, NULL, NULL);
    handle_client(client_connection_socket);
    close(client_connection_socket);
}
```

4. Criar a função `void handle_client(int socket)`, que irá tratar a requisição de um cliente
   (a) Essa função deve receber a requisição do cliente
   (b) A requisição deve ser analisada para verificar se está correta
      - Use a função `sscanf` para extrair o método, objeto e versão da requisição
      - Se não conseguir extrair, você deve responder ao cliente com uma mensagem de erro `400 Bad Request`
   (c) Se o método não for GET ou HEAD, você deve responder ao cliente com uma mensagem de erro `501 Not Implemented`
   (d) Se a versão da requisição não for HTTP/1.1 ou HTTP/1.0 você deve responder ao cliente com uma mensagem de erro `505 HTTP Version Not Supported`
   (e) Se tudo estiver certo, você deve tentar acessar
      i. Se o arquivo solicitado for "/"(use `strcmp` para descobrir), você deve considerar que o nome do arquivo é `index.html` (use `strcpy` para sobrescrever uma string)
      ii. Use a função `open` para abrir o arquivo
         - Se o arquivo não existir, você deve responder ao cliente com uma mensagem de erro `404 Not Found`

iii. Descubra o tamanho do arquivo

```
struct stat file_stat;
fstat(&file, &file_stat);
size_t file_size = file.stat.st_size;
```

iv. Agora que sabe o tamanho, você pode alocar uma string desse tamanho com `malloc` e ler o conteúdo do arquivo com `read`

v. Feche o arquivo com `close`

vi. Descubra o tipo do arquivo

```
char file_type[25];
const char *ext = strrchr(filepath, '.');
if (ext == NULL)
    strcpy(file_type, "application/octet-stream");
else if (strcmp(ext, ".html") == 0)
    strcpy(file_type, "text/html");
else if (strcmp(ext, ".txt") == 0)
    strcpy(file_type, "text/plain");
else if (strcmp(ext, ".jpg") == 0)
    strcpy(file_type, "text/jpeg");
else if (strcmp(ext, ".png") == 0)
    strcpy(file_type, "text/png");
else
    strcpy(file_type, "application/octet-stream");
```

vii. Agora você pode finalmente enviar a reposta ao cliente

- Lembre-se que, se a requisição for um GET, você deve enviar o arquivo e suas informações
- Mas se a requisição for um HEAD, você deve enviar apenas as informações do arquivo (tipo e tamanho)

5. Crie a função `send_error(int socket, const char *status)` para enviar as mensagens de erro

   - A variável `status` contém a mensagem, como `404 Not Found`
   - Use a função `snprintf` para combinar várias strings em uma só
   - Suas respostas de erro devem seguir o formato abaixo. Atente-se que a quebra de linha deve ser um `\r\n` e que há uma linha em branco no final:

     ```
     HTTP/1.1 404 Not Found
     Connection: close
     ```

6. Crie a função `send_response(int socket, const char *status, int header_only, const char *file_type, const char *file_content, size_t file_size)` para enviar as mensagens de sucesso

   - A variável `status` contém a mensagem, como `200 OK`
   - Use a função `snprintf` para combinar várias strings em uma só
   - Suas respostas de erro devem seguir o formato abaixo. Atente-se que a quebra de linha deve ser um `\r\n` e que há uma linha em branco no final:

     ```
     HTTP/1.1 200 OK
     Content-type: text/html
     Content-Length: 25
     ```

```
<h1>Olá, funcionou</h1>
```

- Sugestão: combine as strings do cabeçalho em uma só e envie usando um `send`. Depois, verifique o valor de `header_only` e envie o conteúdo do arquivo usando outro comando `send`

4 Para pensar

Reflita sobre os seguintes pontos:

1. Os sites que acessamos no dia a dia seguem a mesma lógica do nosso cliente e servidor?
2. A aplicação web desenvolvida por você e seus colegas nos anos anteriores segue essa a mesma ideia?
3. O acesso a um banco de dados é parecido com o que fizemos nessa atividade?
4. E a troca de mensagens por WhatsApp?
5. E a comunicação que seu computador faz quando você está jogando um jogo online?
6. E as outras aplicações que você utiliza pela Internet?

5 O trabalho

Essa atividade é parte integrante do conceito avaliativo do primeiro bimestre da disciplina.

A atividade pode ser realizado no máximo em duplas.

O trabalho deve ser realizado em ambiente Linux. Você pode usar uma máquina virtual, se preferir.

6 O que entregar

Você deverá entregar, até o dia 12/maio (conforme calendário), um arquivo *.zip* contendo uma pasta, que por sua vez contém:

- Os dois arquivos de código C do seu servidor e cliente HTTP;
- Um arquivo de texto chamado `README.txt` contendo o nome completo dos integrantes do grupo e quaisquer outras informações sobre o progresso do trabalho que julgarem relevantes;
- Pelo menos dois arquivos de exemplo que o grupo usou para testar o servidor. Os arquivos devem ser de tipos diferentes: por exemplo, um arquivo HTML e outro de TXT;

O arquivo compactado descrito acima deve ser nomeado com as iniciais do nome de cada integrante separados por um hífen. Por exemplo: se o Fulano da Silva fez o trabalho com o João de Souza, o arquivo deve ser nomeado `fs-js.zip`. A pasta, dentro do arquivo compactado, deve ter o mesmo nome, a não ser pela extensão.

A entrega será pelo SUAP.