

Projeto 1

MC833

Prof. Luiz Fernando Bittencourt

RA Nome

157986 Henrique Noronha Facioli

148077 Thiago Silva de Farias

Descrição da Implementação e Funções Utilizadas

Cliente

- Inicialmente, checamos os argumentos utilizados na execução do programa:
 - Caso não seja passado nenhum argumento, imprimimos uma mensagem de erro e retornamos.
 - Caso seja passado somente um argumento, temos que este será o endereço IP e que o número da porta será um valor padrão (12345).
 - Caso sejam passados dois argumentos, o primeiro será o endereço IP e o segundo será o número da porta.
- Em seguida, traduzimos o nome do host para um endereço IP, utilizando a função `"gethostbyname"`, que retorna uma estrutura do tipo `"hostent"` se executada corretamente e, caso contrário, retorna `NULL`. Nesse segundo caso, imprimimos uma mensagem de erro e retornamos.
- Então, criamos um socket utilizando a função `"socket(AF_INET, SOCK_STREAM, 0)"` e checamos se a criação foi feita corretamente. Em caso negativo, imprimimos uma mensagem de erro e retornamos. Em caso positivo, inicializamos o socket com a função `"bzero((char *)&socket_address, sizeof(socket_address))"` e então o preenchemos com as informações necessárias.
- Por se tratar do cliente, devemos agora realizar a criação de um socket ativo, utilizando a função `"connect(int sockfd, const struct sockaddr *servaddr, int addrlen)"`, que o cliente usa para iniciar conexão com o servidor. Em caso de falha, imprimimos uma mensagem de erro e retornamos.
- Requisitamos então do usuário uma mensagem, que será enviada ao servidor, utilizando a função `"fgets"`, e, em seguida, tentamos enviar esta mensagem ao servidor, com o uso da função `"send(int sockfd, const void`

**msg, int len, int flags*)". Em caso de problemas no envio da mensagem, imprimimos uma mensagem de erro e retornamos.

- Por fim, tentamos receber uma resposta do servidor, através da função *"recv(int sockfd, void *buf, int len, int flags)"*. Caso não seja recebida uma resposta, imprimimos uma mensagem de erro e retornamos. Caso contrário, imprimimos a mensagem recebida.

Servidor

- Inicialmente, checamos os argumentos utilizados na execução do programa:
 - Caso não seja passado nenhum argumento, temos que o número da porta será um valor padrão (12345).
 - Caso seja passado somente um argumento, temos que este será o endereço número da porta utilizado. Caso ocorra algum problema com o valor de entrada, imprimimos uma mensagem de erro e retornamos.
- Então, criamos um socket utilizando a função *"socket(AF_INET, SOCK_STREAM, 0)"* e checamos se a criação foi feita corretamente. Em caso negativo, imprimimos uma mensagem de erro e retornamos. Em caso positivo, inicializamos o socket com a função *"bzero((char *)&socket_address, sizeof(socket_address))"* e então ele é preenchido com as informações necessárias.
- Em seguida, associamos o socket ao descritor através da função *"bind(int sockfd, struct sockaddr *addr, int addrlen)"*, e, caso ocorra algum problema, imprimimos uma mensagem de erro e retornamos.
- Por se tratar do servidor, devemos agora criar a escuta do socket para aceitar conexões. Isso é feito com o uso da função *"listen (int sockfd, (struct sockaddr) *myaddr, int backlog)"*. Em caso de falha, imprimimos uma mensagem de erro e retornamos.
- Por fim, devemos aguardar e aceitar as conexões requisitadas. Para isso, realizamos os seguintes passos em loop:
 - Primeiramente utilizamos a função *"accept (int sockfd, (struct sockaddr) *cliaddr, int *socklen)"*, para aceitar a conexão com um cliente. Caso ocorra algum problema, imprimimos uma mensagem de erro.
 - Em seguida, tentamos receber uma requisição do cliente através da função *"recv(int sockfd, void *buf, int len, int flags)"*. Caso a mensagem seja recebida com sucesso, imprimimos ela e o seu remetente.
 - Então, utilizamos a função *"send(int sockfd, const void *msg, int len, int flags)"* para enviar um eco ao cliente, de forma a avisá-lo que a mensagem foi recebida com sucesso.

Testes realizados para validar a implementação

A seguir apresentamos imagens representando uma troca de mensagens entre cliente e servidor, de modo a validar a nossa implementação. No primeiro par de imagens, não especificamos a porta utilizada (ou seja, usamos a porta 12345). Já no segundo par de imagens, utilizamos uma porta específica (no caso, 2048). Em ambos os casos, o cliente passa como argumento “localhost” como servidor alvo.

Na imagem abaixo, temos o cliente enviando uma mensagem ao servidor, e, em seguida, recebendo um “echo”.

```
henrique@nf ~$ ./client localhost
Escreva a mensagem a ser enviada
tsdfarias é maneiro
tsdfarias é maneiro
henrique@nf ~$
```

10053 10:42:00

10055 10:42:10

Agora, temos uma imagem mostrando a mensagem proveniente do cliente, recebida pelo servidor, que então imprime o identificador do cliente, além da mensagem recebida.

```
henrique@nf ~$ ./server
Running Server on 12345
Message received
From: 16777343
Message:
tsdfarias é maneiro
```

Por fim, temos duas imagens representando o mesmo que as anteriores (e na mesma ordem), porém, especificando o número da porta como 2048.

```
henrique@nf ~$ ./client localhost 2048
Escreva a mensagem a ser enviada
lces também!
lces também!
henrique@nf ~$
```

10056 10:45:31

```
henrique@nf ~$ ./server 2048
Running Server on 2048
Message received
From: 16777343
Message:
lces também!
```