Lauro Cruz e Souza - 156175

Pedro Emílio Machado de Brito - 137264

Projeto 1

server.c

O servidor foi implementado de forma a esperar em loop por conexões, e também em um loop, ler uma linha de texto, imprimi-la, mandar eco, e tentar ler mais texto. Quando não é possível ler mais texto, a conexão é fechada e o servidor tenta aceitar outra.

Todas as funções que tratam da conexão com a rede tem seus códigos de erro verificados e tratados, se necessário.

Algumas das mensagens de erro que obtivemos:

`ERROR: Unable to bind socket: Permission denied:`

Causado quando a porta escolhida era baixa, ou seja, privilegiada, e o binário de servidor era executado sem privilégios de root.

`ERROR: Unable to bind socket: Address already in use:`

Causado quando já havia uma instância do servidor presente escutando naquele endereço e porta.

client.c

O cliente foi implementado de forma a se conectar ao servidor, e então entrar em um loop de enviar uma linha de texto, e esperar pelo eco. O cliente para de ler texto e fecha a conexão ao receber o sinal CTRL-D (EOF), e então sai.

Algumas das mensagens de erro que obtivemos:

`ERROR: Unable to resolve hostname:`

Ao tentar se conectar a um hostname inválido.

`ERROR: Unable to connect to server: Connection refused:`

Ao se tentar conectar a uma porta onde não há um servidor escutando.

Funções utilizadas nos programas:

`bzero`: limpa uma string, escrevendo caracteres nulos (`\0`) até o tamanho especificado.

`htonl`, `htons`: converte ordem de bytes, de ordem de host pra ordem de rede.

`gethostbyname`: retorna um ponteiro para um `struct hostent`, que contém informações sobre o host resolvido.

`socket`: cria um "endpoint" ainda não associado a um endereço IP e porta TCP

`bind`: associa o socket anteriormente criado ao endereço e porta desejados.

`listen`: começa a escutar por conexões, com limite de conexões pendentes.

`accept`: aceita uma conexão pendente, retornando o socket associado.

`connect`: tenta se conectar com o host remoto usando um socket anteriormente criado.

`recv`: recebe dados no socket.

`send`: envia dados no socket.

`close`: fecha o socket, terminando a conexão.

Um exemplo de sessão:

Em um terminal, rodamos:

`$ ./server`

e em outro

`$ ./client localhost`

estabelecendo a conexão. A porta usada é implicitamente 12345, pelas constantes definidas em cada arquivo fonte.

Várias linhas de texto são digitadas no console com o cliente aberto:

```
$ ./client localhost
oi
oi
teste
teste
batata
batata
the quick brown fox jumps over the lazy dog
the quick brown fox jumps over the lazy dog
```

Nesse ponto a combinação CTRL-D é pressionada e o cliente termina. É possivel ler as linhas digitadas e logo em seguida, o eco do servidor.

Enquanto isso, no terminal onde está aberto o servidor:

```
$ ./server
oi
teste
batata
the quick brown fox jumps over the lazy dog
```

Mesmo o cliente tendo terminado, o servidor continua ativo, esperando, outras conexões. Faremos isso, abrindo outra sessão do cliente:

```
$ ./client localhost
segunda sessão
segunda sessão
mais linhas
mais linhas
ainda mais linhas
ainda mais linhas
```

E no servidor, o relato completo de todas as linhas já recebidas:

```
$ ./server
oi
teste
batata
the quick brown fox jumps over the lazy dog
segunda sessão
mais linhas
ainda mais linhas
```

Em seguida, terminamos o servidor usando CTRL-C.