



ENGENHARIA DA COMPUTAÇÃO
REDES DE COMPUTADORES A

ATIVIDADE 2

EQUIPE:

Agostinho Sanches de Araújo -----	RA: 16507915
Evandro Douglas Capovilla Junior -----	RA: 16023905
Lucas Tenani Felix Martins -----	RA: 16105744
Pedro Andrade Caccavaro -----	RA: 16124679

19/03/2019



SUMÁRIO

INTRODUÇÃO.....	03
OBJETIVO.....	03
DESCRIÇÃO.....	04
1. CADASTRAR MENSAGEM.....	04
2. LER MENSAGENS.....	05
3. APAGAR MENSAGENS.....	05
4. SAIR DA APLICAÇÃO.....	05
FUNÇÕES.....	06
OBSERVAÇÕES.....	07
RESULTADO.....	08
CONCLUSÃO.....	09

INTRODUÇÃO

Transmission Control Protocol, conhecido como TCP, é um dos protocolos mais utilizados da camada de transporte do modelo TCP/IP. O TCP é orientado a conexão, ou seja, é necessário estabelecer uma conexão prévia entre as máquinas antes que ocorra o envio de pacotes, denominado como *Three Way Handshake*. Dessa maneira é garantido que todos os pacotes serão entregues do emissor ao receptor, através de inúmeras confirmações e comunicações entre as máquinas e, no caso de perda de pacote, é enviado um pedido de retransmissão.

OBJETIVO

A atividade tem como principal objetivo demonstrar a interação de dois programas diferentes se comunicando através da rede. Para efetuar a comunicação via rede, foi utilizado um cliente TCP e um servidor TCP.

DESCRIÇÃO

A atividade tem início com a execução do programa servidor, no qual é passado um parâmetro correspondente a porta que será utilizada. Assim, um *socket* é criado e o servidor é setado com as devidas informações.

Após setar as informações necessárias para o funcionamento do servidor, é feita uma conexão (*bind*) entre o *socket* criado e o próprio servidor. Com o êxito do *bind*, uma chamada *listen* é criada promovendo uma lista de mensagens. O retorno dessa chamada será 0, quando o *listen* for criado de maneira certa, ou -1, quando a chamada ocasiona em algum erro. Em sequência, a função *accept* é chamada para entrar em contato com o programa cliente, fazendo com que o programa servidor fique em uma espera bloqueante até estabelecer uma conexão. Essa função retorna um outro *socket*, onde está vinculado o programa servidor e o programa cliente, quando têm êxito, ou -1 quando ocorre alguma falha na criação desse *socket*.

Com o programa servidor em espera bloqueante, o programa cliente é executado passando como parâmetro o endereço IP (endereço local da máquina - 127.0.0.1) ou o nome apropriado que indica o endereço IP local (localhost) e a mesma porta que foi utilizada para o programa servidor. Após a execução do programa cliente, um *socket* é criado e o servidor cliente é configurado do mesmo formato que o programa servidor. Estabelecido o *socket*, a função *connect* é chamada para efetuar a conexão entre o programa cliente e o programa servidor fazendo com que o programa servidor não fique mais em espera bloqueante e espere a mensagem que programa cliente irá mandar. O retorno dessa função será 0, quando a conexão for estabelecida com sucesso, ou -1 caso ocorra algum erro.

Um menu de opções é mostrado para o usuário, após a conexão ser estabelecida, sendo elas:

1. CADASTRAR MENSAGEM

O programa cliente pedirá o nome do usuário (até 19 caracteres) e a mensagem que deseja enviar (até 79 caracteres). Esses dados são colocados dentro de uma *struct* chamada mensagem e enviadas para o programa servidor por meio da função *send*. Após o envio, o programa servidor deve guardar as informações (usuário e a mensagem).

O programa cliente fica em espera bloqueante até o programa servidor enviar uma resposta sobre o estado da requisição solicitada (positivo ou negativo).

O servidor deve permitir alocar no máximo 10 mensagens.



2. LER MENSAGENS

O programa cliente fica em espera bloqueante até o programa servidor enviar todas as mensagens que foram cadastradas. É mostrado como resultado no programa cliente o número de mensagens que foram lidas e as informações recebidas pelo programa servidor.

3. APAGAR MENSAGENS

O programa cliente pedirá o nome do usuário. O nome será mandado para o programa servidor e todas as mensagens que tenham sido enviadas por esse usuário serão apagadas.

O programa cliente fica em espera bloqueante até o programa servidor enviar uma resposta sobre o estado da requisição solicitada (positivo ou negativo).

4. SAIR DA APLICAÇÃO

O programa cliente é encerrado e o programa servidor finaliza a conexão com o mesmo, mas permanece executando esperando por outra conexão.

FUNÇÕES UTILIZADAS

Abaixo, todas as funções que foram apresentadas para a realização da atividade com as eventuais explicações sobre seus funcionamentos, parâmetros que foram utilizados e seus retornos.

- **htons(uint16_t hostshort);**
Converte o inteiro não sinalizado *hostshort* de bytes em ordem de host para bytes em ordem de rede.
- **int socket(int domain, int type, int protocol);**
A função *socket* cria um terminal de comunicação.
- **int listen(int sockfd, int backlog);**
Cria uma fila de requisições de conexões com o *socket sockfd* que serão utilizadas através da função *accept*, definido pelo tamanho passado pelo parâmetro *backlog*.
- **int accept(int socket, struct sockaddr *restrict address, socklen_t *restrict address_len);**
Extraí a primeira conexão da fila de conexões pendentes e cria um novo *socket* com o mesmo protocolo do *socket* especificado.
- **in_addr_t inet_addr(const char *cp);**
Converte o endereço IP números-e-pontos em dados binários na ordem de bytes da rede.
- **void *memset(void *s, int c, size_t n);**
Preenche os *n* primeiros bytes da área de memória apontada por *s* com a constante *c*.
- **strcmp(const char *s1, const char *s2);**
Compara duas strings *s1* e *s2*, o valor de retorno é negativo para *s1* maior que *s2*, nulo para strings iguais e positivo para *s2* maior que *s1*.
- **void *memcpy(void *dest, const void *src, size_t n);**
Copia *n* bytes da área de memória *src* para a área de memória *dest*.

- **ssize_t send(int sockfd, const void *buf, size_t len, int flags);**
Transmite uma mensagem *buf*, de tamanho *len*, utilizando o *socket* *sockfd*.
- **ssize_t recv(int sockfd, void *buf, size_t len, int flags);**
Recebe uma mensagem *buf*, de tamanho *len*, através do *socket* *sockfd*.
- **int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);**
Atribui o endereço especificado no *socket* pelo arquivos descritor.
- **int close(int fd);**
Desaloca da memória o *socket* *fd*.
- **void __fpurge(FILE *stream);**
Limpa o buffer de *stream*.
- **int scanf(const char *format, ...);**
Lê o input de com o formato passado em **format*, utilizado no código como `%[^\n]` para ler até que encontre `'\0'`.
- **FILE *popen(const char *command, const char *type);**
Executa um processo solicitado invocando o shell.

OBSERVAÇÕES

A escolha da utilização de *structs* foi devido à facilidade de enviar os dados para o servidor, Usuário e Mensagem.

Não houve validação de tamanho do nome de usuário ou mensagem.

RESULTADOS

```
evandro@evandro-capo: ~/Desktop/trabalho
File Edit View Search Terminal Help
evandro@evandro-capo:~/Desktop/trabalho$ ./cliente 127.0.0.1 3002
Digite o seu opção: 1
Digite seu nome: Evandro
Digite seu texto: Olá tudo bem ?
Mensagem enviada ao servidor
Mensagem recebida do servidor: Incluso com sucesso!
Digite o seu opção: 1
Digite seu nome: Pedro
Digite seu texto: Oi, sim e com você ?
Mensagem enviada ao servidor
Mensagem recebida do servidor: Incluso com sucesso!
Digite o seu opção: 1
Digite seu nome: Evandro
Digite seu texto: Eu tambem estou, preciso sair agora ! Tchau !
Mensagem enviada ao servidor
Mensagem recebida do servidor: Incluso com sucesso!
Digite o seu opção: 1
Digite seu nome: Pedro
Digite seu texto: Tudo bem ! Tchau !
Mensagem enviada ao servidor
Mensagem recebida do servidor: Incluso com sucesso!
Digite o seu opção: 2
Mensagem 1
Nome: Evandro
Mensagem: Olá tudo bem ?
Mensagem 2
Nome: Pedro
Mensagem: Oi, sim e com você ?
Mensagem 3
Nome: Evandro
Mensagem: Eu tambem estou, preciso sair agora ! Tchau !
Mensagem 4
Nome: Pedro
Mensagem: Tudo bem ! Tchau !
```

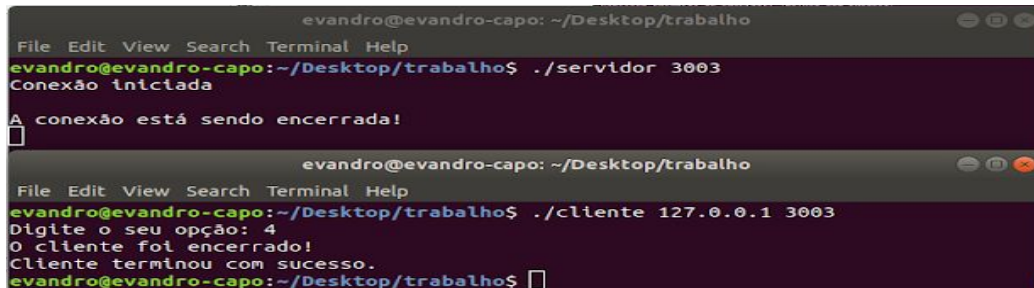
```
evandro@evandro-capo: ~/Desktop/trabalho
File Edit View Search Terminal Help
evandro@evandro-capo:~/Desktop/trabalho$ ./servidor 3002
Conexão iniciada
Mensagem do Cliente: Evandro
Mensagem recebida do cliente: Olá tudo bem ?
Incluso com sucesso!
Mensagem do Cliente: Pedro
Mensagem recebida do cliente: Oi, sim e com você ?
Incluso com sucesso!
Mensagem do Cliente: Evandro
Mensagem recebida do cliente: Eu tambem estou, preciso sair agora ! Tchau !
Incluso com sucesso!
Mensagem do Cliente: Pedro
Mensagem recebida do cliente: Tudo bem ! Tchau !
Incluso com sucesso!
```

As fotos acima ilustram o funcionamento dos programas cliente e servidor utilizando a opções 1 para cadastrar mensagem, e opção 2 para ler mensagem.

```
evandro@evandro-capo: ~/Desktop/trabalho
File Edit View Search Terminal Help
evandro@evandro-capo:~/Desktop/trabalho$ ./servidor 3002
Conexão iniciada
As mensagens foram apagadas
[]

evandro@evandro-capo: ~/Desktop/trabalho
File Edit View Search Terminal Help
evandro@evandro-capo:~/Desktop/trabalho$ ./cliente 127.0.0.1 3002
Digite o seu opção: 3
Removendo as mensagens do servidor!
Mensagem recebida do servidor: As mensagens foram apagadas
Digite o seu opção: []
```

A foto acima ilustra os programas cliente e servidor utilizando a opção 3 para remover mensagem.



```
evandro@evandro-capo: ~/Desktop/trabalho
File Edit View Search Terminal Help
evandro@evandro-capo:~/Desktop/trabalho$ ./servidor 3003
Conexão iniciada
A conexão está sendo encerrada!
█

evandro@evandro-capo: ~/Desktop/trabalho
File Edit View Search Terminal Help
evandro@evandro-capo:~/Desktop/trabalho$ ./cliente 127.0.0.1 3003
Digite o seu opção: 4
O cliente foi encerrado!
Cliente terminou com sucesso.
evandro@evandro-capo:~/Desktop/trabalho$ █
```

A foto acima mostra o encerramento do servidor através da opção 4.

CONCLUSÃO

O foco durante a realização deste trabalho fora a melhor compreensão acerca do protocolo TCP, estudando suas definições e aplicações para poder implementar o sistema Cliente/Servidor. Agora, tendo implementado o protocolo TCP pudemos concluir, no sentido comparativo, que entre os dois protocolos o TCP garante com mais certeza a chegada dos pacotes enviados, pois há a validação do servidor para cada pacote enviado, além de haver um fila de requisições que vão armazenando as requisições enviadas pelo cliente.