

Trabalho prático 2 - Documentação

Introdução

Este trabalho consiste na implementação de um sistema de *chat* utilizando o protocolo IRC-2 na linguagem C. Devem ser implementados um programa servidor e outro cliente, onde eles se comunicariam utilizando através da rede usando o protocolo UDP/IP. O servidor deve ser capaz de se comunicar com varios clientes ao mesmo tempo, sendo o caminho de transporte de mensagens entre clientes. O sistema também deve garantir a entrega das mensagens. Como essa garantia não é fornecida pelo protocolo UDP, ela deve ser implementada no proprio sistema.

Protocolo IRC-2

O protocolo IRC-2 é baseado em texto, ou seja, os comandos são strings de caracteres ASCII. Além disso, todos os comandos são digitados com caracteres maiúsculos. Os seguintes comandos são suportados:

- **NICK <string>**: Define o nome de usuário que será utilizado pelo cliente durante uma conexão. Este deve ser o primeiro comando enviado pelo usuário. O apelido não deve conter caracteres de espaço, e tem um tamanho limitado a 16 caracteres.
- **POST <string>**: É empregado para enviar mensagens no sentido cliente -> servidor. As mensagens são texto livre, de no máximo 500 caracteres, sempre terminados por um caractere “\n”, que é contabilizado dentro do limite de 500 caracteres.
- **NEW <user> <string>**: É empregado para enviar mensagens no sentido servidor cliente. As mensagens são texto livre, de no máximo 500 caracteres, sempre terminados por um caractere “\n”, que é contabilizado dentro do limite de 500 caracteres. A string <user> identifica o usuário que enviou a mensagem.

- **MUTE <user>**: Empregado no sentido cliente -> servidor, indica ao servidor que ele deve parar de enviar as mensagens de um dado cliente, identificado pelo apelido <user>, para o usuário.
- **UNMUTE <user>**: Empregado no sentido cliente -> servidor, indica ao servidor que ele deve voltar a enviar as mensagens de um dado cliente, identificado pelo apelido <user>, para o usuário.
- **CLOSE**: Comando enviado pelo cliente, termina a conexão ao servidor. Quando o cliente enviar o **CLOSE** o servidor deverá limpar da sua memória a lista de usuários em mudo. Assim quando o usuário logar novamente, ele irá receber as mensagens de todos os usuários, mesmo se ele os colocou em mudo em uma conexão anterior.

Vale ressaltar que todos os comandos são terminados por um “\n”, e que os parâmetros de um comando são sempre separados por espaços.

Decisões de projeto

Algumas decisões de projeto foram tomadas de modo a resolver os problemas propostos na especificação.

O primeiro problema a ser resolvido era da comunicação confiável. Como protocolo UDP não garante o recebimento de mensagens enviados, foi necessário implementar um sistema próprio de confirmação de mensagens. A cada mensagem recebida, um pacote de ACK é enviado para o remetente da mensagem. Quando uma mensagem é enviada, o programa aguarda por uma mensagem de ACK do outro ponto da conexão. O tempo de aguardo para a mensagem de ACK é definido pela variável `TIMEOUT_INIT`. Caso um pacote de ACK não seja recebido durante esse tempo, a mensagem é retransmitida, e a resposta é aguardada pelo dobro o tempo anterior. O processo continua até a mensagem de ACK ser recebida ou o tempo de espera se tornar maior que a variável `TIMEOUT_LIMIT`.

Os tempos definidos nas variáveis `TIMEOUT_INIT` e `TIMEOUT_LIMIT` são de 1s e 30s respectivamente. Para fins de simplicidade, a mensagem ACK é enviada sozinha, ou seja, o técnica de piggyback não foi implementada.

Os outros problemas a serem resolvidos são relacionados ao programa cliente ou servidor

Servidor

O servidor deve ser capaz de se comunicar com varios clientes ao mesmo tempo. Felizmente, como o protocolo de transporte usado é o UDP, a comunicação é feita sem conexão. Assim, ao receber a mensagem, o servidor determina o seu remetente e a processa de acordo.

Cada comando enviado ao servidor deve ser respondido, de modo a informar o sucesso ou insucesso do mesmo. As respostas do servidor aos comandos estão compilados na tabela a seguir

RESPOSTA	DESCRIÇÃO	COMANDOS APLICAVEIS
COMMAND success	O comando foi executado com sucesso	Todos
COMMAND not registered	O usuário não está registrado, é necessário executar o comando NICK	Todos exceto NICK
COMMAND no user	O usuário informado não existe	MUTE UNMUTE
MUTE already muted	O usuario ja foi colocado no mudo	MUTE
UNMUTE not muted	O usuario não esta no mudo	UNMUTE
ERROR invalid command	Resposta padrão para mensagens fora do protocolo IRC-2	n/a

Note que COMMAND é substituído pelo comando do qual a resposta é referente.

Cliente

O principal desafio do programa cliente é que mensagens podem ser recebidas a qualquer momento, quando um outro cliente envia uma mensagem por exemplo, porem o programa deve também esperar por *input* do usuário. Para evitar o uso de multiplas threads, o comando select() do unix foi usado. O comando select é um comando bloqueante da bibliotecaunistd.h que recebe como entrada um conjunto de descritores de arquivo, e retorna quando um desses descritores esta pronto para ser lido. Deste modo o comando select é executado passando o conjunto do descritores de arquivo do socket e da entrada padrão. Quando o select() retorna, o sistema determina se foi devido a uma mensagem no socket ou um input do usuário e executa a ação de acordo

Outro desafio é que o programa deve ser capaz de reconhecer os comandos do usuário das mensagens q ele deseja enviar. Desta forma, o programa cliente trata qualquer input do usuário começado com o caractere '/' como um comando, e todo o resto como mensagem. Os comandos disponíveis ao usuário são /mute /unmute e /close, equivalente aos comandos do protocolo.

Organização do código

O código está organizado em quatro módulos principais: O main, socket, client, server.

O módulo main são os dois arquivos main dos dois programas. O clientmain.c se refere ao main do programa cliente e o servermain.c se refere ao main do programa servidor.

O módulo socket implementa a base da comunicação. Ele implementa usando sockets a comunicação entre aplicações, além de implementar o mecanismo de confirmação de mensagens. O módulo está dividido em socket.h e socket.c

O módulo server possui as rotinas para a execução dos comandos do protocolo, assim como as estruturas que armazenam os clientes conectados e suas informações

O módulo client possui as rotinas para o envio dos comandos para o servidor, bem como o tratamento do input do usuário.

Além dos módulos também foi fornecido um arquivo Makefile para a compilação utilizando a ferramenta make.

O funcionamento básico dos dois programas está descrito a seguir:

Servidor

1. Inicializa as estruturas necessárias
2. Abre o socket e dá bind na porta especificada
3. Aguarda por mensagens
4. Processa a mensagem recebida, enviando as respostas e mensagens para os respectivos clientes
5. Retorna ao passo 3

Cliente

1. Cria o socket de rede
2. Envia o comando nick ao servidor especificado, passando o nome de usuário especificado
3. Usa o select() no stdin e no sockfd
4. Determina o file descriptor a ser lido
5. Processa o input do usuário ou a mensagem recebida pelo servidor

6. Determina se deve continuar executando pelo comando anterior executado anteriormente
7. Se sim retorna ao passo 3

Testes

Os testes foram feitos em tres tipos de rede: localhost, LAN e WAN. O objetivo foi analisar o comportamento do programa nas tres situações e testar o funcionamento do sistema de chat nas mais variadas configurações de rede.

Em todas as configurações de rede foram conectados 2 ou mais clientes em um servidor e mensagens foram trocadas entre eles, assim como comandos de mute e unmute para determinar se os clientes estavam sendo colocados no mudo corretamente.

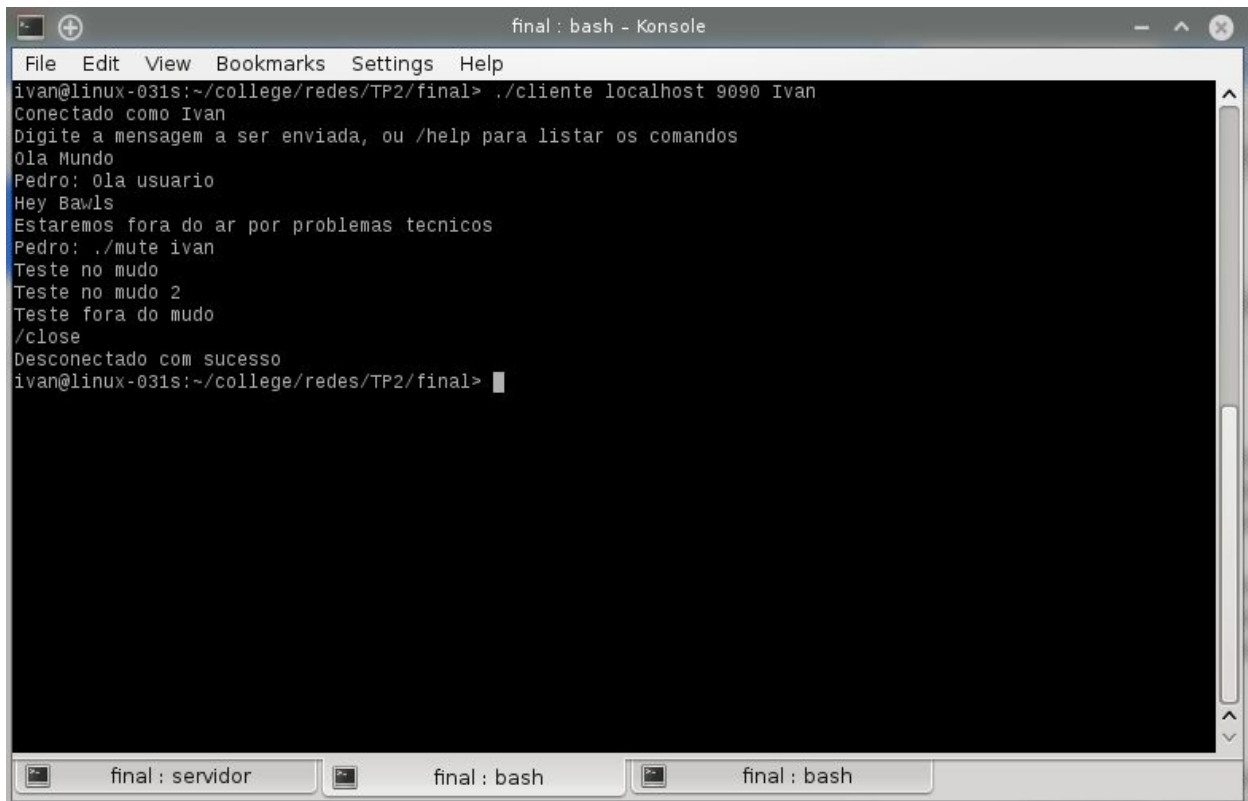
Localhost

O primeiro teste foi realizado na mesma máquina. Foi executado um processo de servidor, aguardando conexões na porta 9090, e três processos clientes.

Em um dos clientes a conexão foi feita usando o nome localhost, enquanto no outro foi feito usando o endereço local IPv6 ::1.

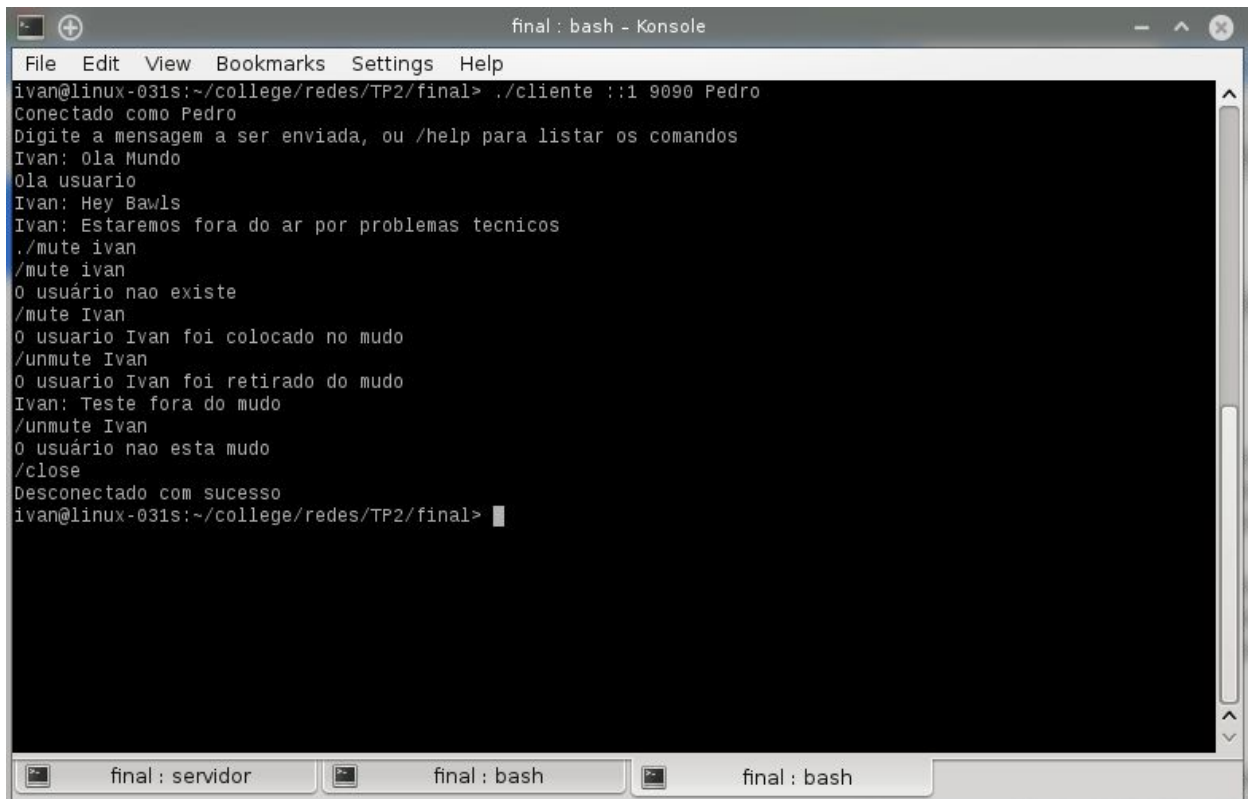
```
final : servidor - Konsole
File Edit View Bookmarks Settings Help
ivan@linux-031s:~/college/redes/TP2/final> ./servidor 9090
Received <::1:53666> => NICK Ivan
Received <::1:37046> => NICK Pedro
Received <::1:53666> => POST Ola Mundo
Received <::1:37046> => POST Ola usuario
Received <::1:53666> => POST Hey Bawls
Received <::1:53666> => POST Estaremos fora do ar por problemas tecnicos
Received <::1:37046> => POST ./mute ivan
Received <::1:37046> => MUTE ivan
Received <::1:37046> => MUTE Ivan
Received <::1:53666> => POST Teste no mudo
Received <::1:53666> => POST Teste no mudo 2
Received <::1:37046> => UNMUTE Ivan
Received <::1:53666> => POST Teste fora do mudo
Received <::1:37046> => UNMUTE Ivan
Received <::1:37046> => CLOSE
Received <::1:53666> => CLOSE
```

Servidor



```
final : bash - Konsole
File Edit View Bookmarks Settings Help
ivan@linux-031s:~/college/redes/TP2/final> ./cliente localhost 9090 Ivan
Conectado como Ivan
Digite a mensagem a ser enviada, ou /help para listar os comandos
Ola Mundo
Pedro: ola usuario
Hey Bawls
Estaremos fora do ar por problemas tecnicos
Pedro: ./mute ivan
Teste no mudo
Teste no mudo 2
Teste fora do mudo
/close
Desconectado com sucesso
ivan@linux-031s:~/college/redes/TP2/final>
```

Cliente 1

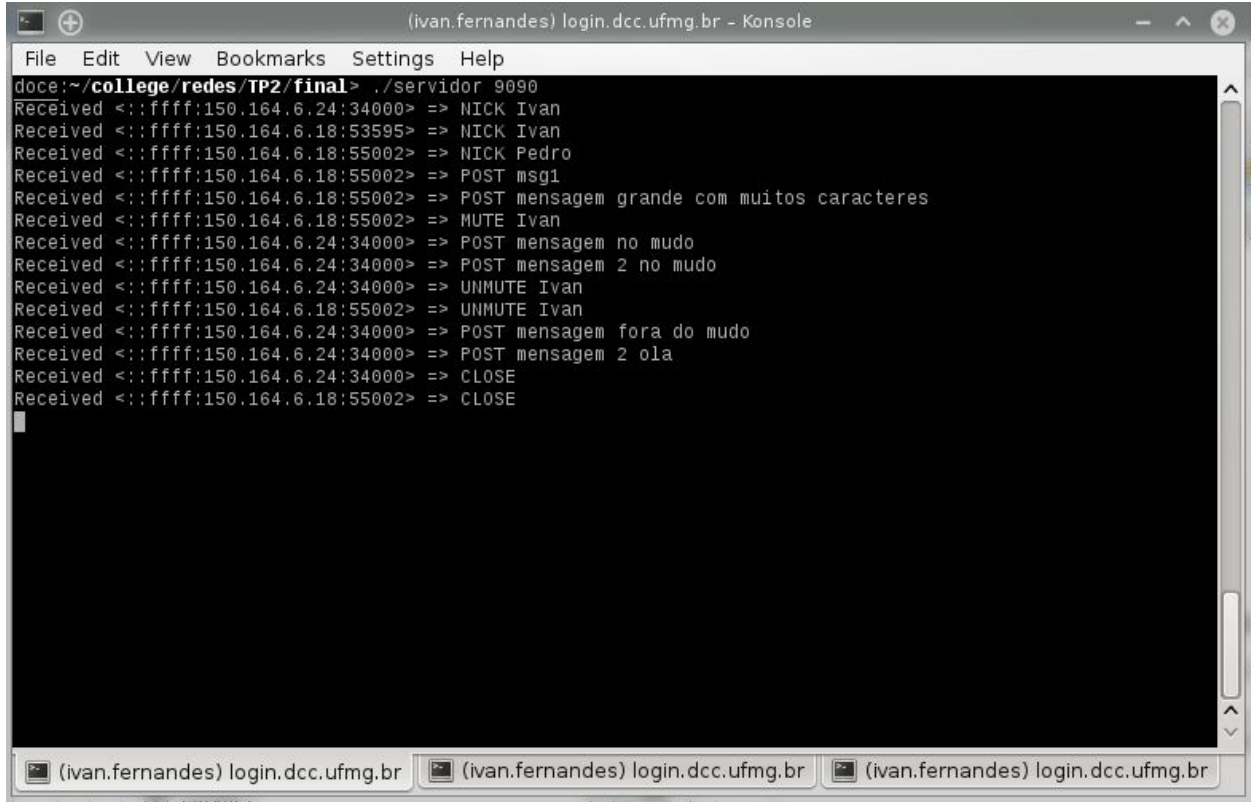


```
final : bash - Konsole
File Edit View Bookmarks Settings Help
ivan@linux-031s:~/college/redes/TP2/final> ./cliente :1 9090 Pedro
Conectado como Pedro
Digite a mensagem a ser enviada, ou /help para listar os comandos
Ivan: ola Mundo
ola usuario
Ivan: Hey Bawls
Ivan: Estaremos fora do ar por problemas tecnicos
./mute ivan
/mute ivan
O usuário nao existe
/mute Ivan
O usuário Ivan foi colocado no mudo
/unmute Ivan
O usuário Ivan foi retirado do mudo
Ivan: Teste fora do mudo
/unmute Ivan
O usuário nao esta mudo
/close
Desconectado com sucesso
ivan@linux-031s:~/college/redes/TP2/final>
```

Cliente 2

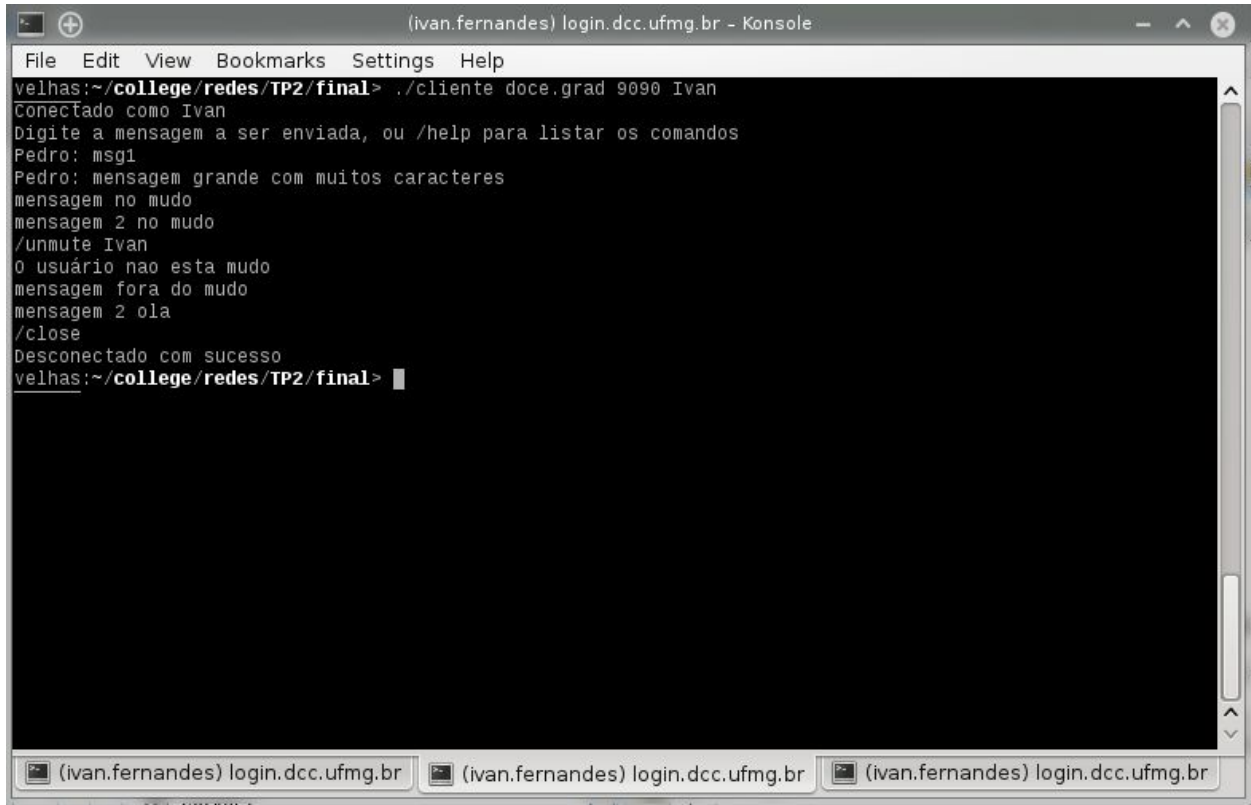
Rede LAN

Para os testes em rede lan os programas foram executados nas maquinas linux do laboratório do dcc. O servidor foi executado na maquina doce.grad, um dos clientes na maquina velhas.grad e o outro cliente na maquina claro.grad. O servidor ouvia a porta 9090



```
(ivan.fernandes) login.dcc.ufmg.br - Konsole
File Edit View Bookmarks Settings Help
doce:~/college/redes/TP2/final> ./servidor 9090
Received <::ffff:150.164.6.24:34000> => NICK Ivan
Received <::ffff:150.164.6.18:53595> => NICK Ivan
Received <::ffff:150.164.6.18:55002> => NICK Pedro
Received <::ffff:150.164.6.18:55002> => POST msg1
Received <::ffff:150.164.6.18:55002> => POST mensagem grande com muitos caracteres
Received <::ffff:150.164.6.18:55002> => MUTE Ivan
Received <::ffff:150.164.6.24:34000> => POST mensagem no mudo
Received <::ffff:150.164.6.24:34000> => POST mensagem 2 no mudo
Received <::ffff:150.164.6.24:34000> => UNMUTE Ivan
Received <::ffff:150.164.6.18:55002> => UNMUTE Ivan
Received <::ffff:150.164.6.24:34000> => POST mensagem fora do mudo
Received <::ffff:150.164.6.24:34000> => POST mensagem 2 ola
Received <::ffff:150.164.6.24:34000> => CLOSE
Received <::ffff:150.164.6.18:55002> => CLOSE
```

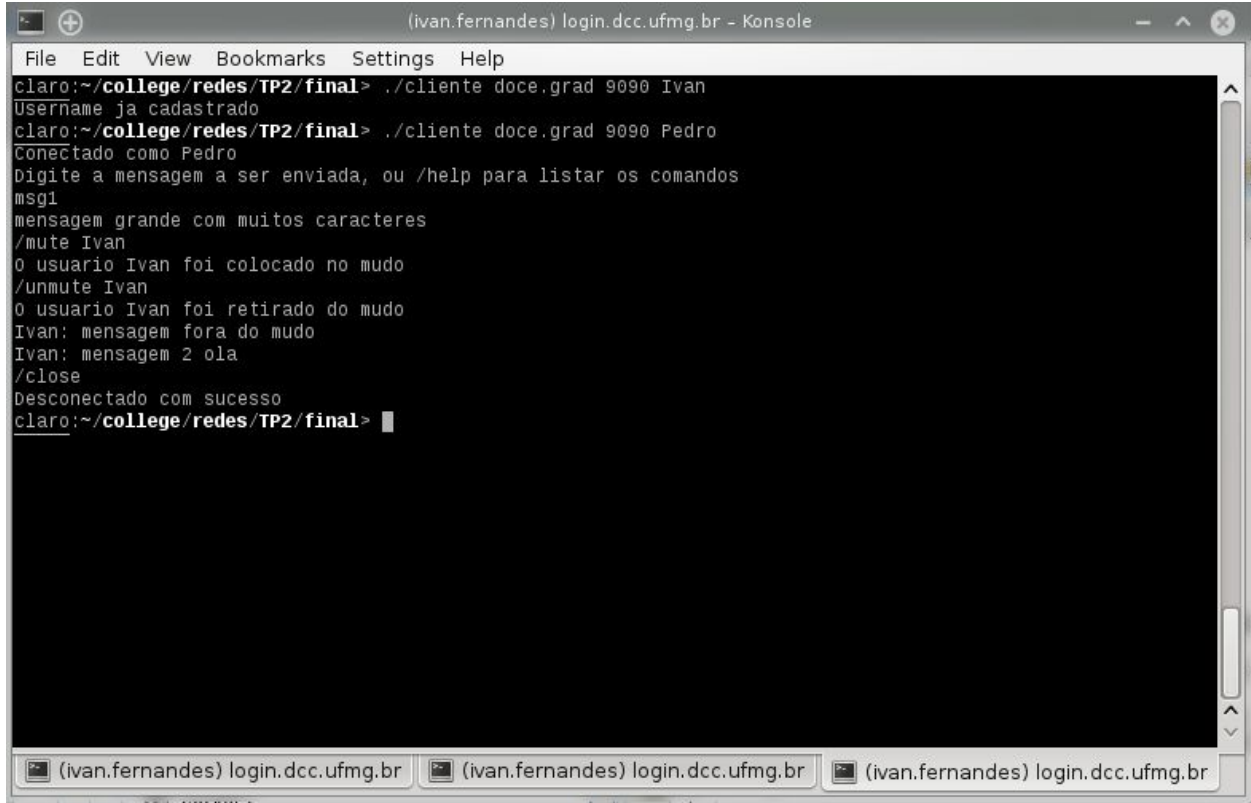
Servidor



A terminal window titled "(ivan.fernandes) login.dcc.ufmg.br - Konsole". The window has a menu bar with "File", "Edit", "View", "Bookmarks", "Settings", and "Help". The terminal output shows a user named "velhas" logging in as "Ivan" using the command ". /cliente doce.grad 9090 Ivan". The user is connected and receives a list of commands: "msg1", "mensagem grande com muitos caracteres", "mensagem no mudo", "mensagem 2 no mudo", "/unmute Ivan", "O usuário nao esta mudo", "mensagem fora do mudo", "mensagem 2 ola", and "/close". The user enters "/close" and is disconnected successfully. The prompt returns to "velhas:~/college/redes/TP2/final>".

```
(ivan.fernandes) login.dcc.ufmg.br - Konsole
File Edit View Bookmarks Settings Help
velhas:~/college/redes/TP2/final> ./cliente doce.grad 9090 Ivan
Conectado como Ivan
Digite a mensagem a ser enviada, ou /help para listar os comandos
Pedro: msg1
Pedro: mensagem grande com muitos caracteres
mensagem no mudo
mensagem 2 no mudo
/unmute Ivan
O usuário nao esta mudo
mensagem fora do mudo
mensagem 2 ola
/close
Desconectado com sucesso
velhas:~/college/redes/TP2/final> █
```

Cliente 1



A terminal window titled "(ivan.fernandes) login.dcc.ufmg.br - Konsole". The window has a menu bar with "File", "Edit", "View", "Bookmarks", "Settings", and "Help". The terminal output shows a user named "claro" logging in as "Ivan" using the command ". /cliente doce.grad 9090 Ivan". The user is connected and receives a list of commands: "msg1", "mensagem grande com muitos caracteres", "/mute Ivan", "O usuário Ivan foi colocado no mudo", "/unmute Ivan", "O usuário Ivan foi retirado do mudo", "Ivan: mensagem fora do mudo", "Ivan: mensagem 2 ola", and "/close". The user enters "/close" and is disconnected successfully. The prompt returns to "claro:~/college/redes/TP2/final>".

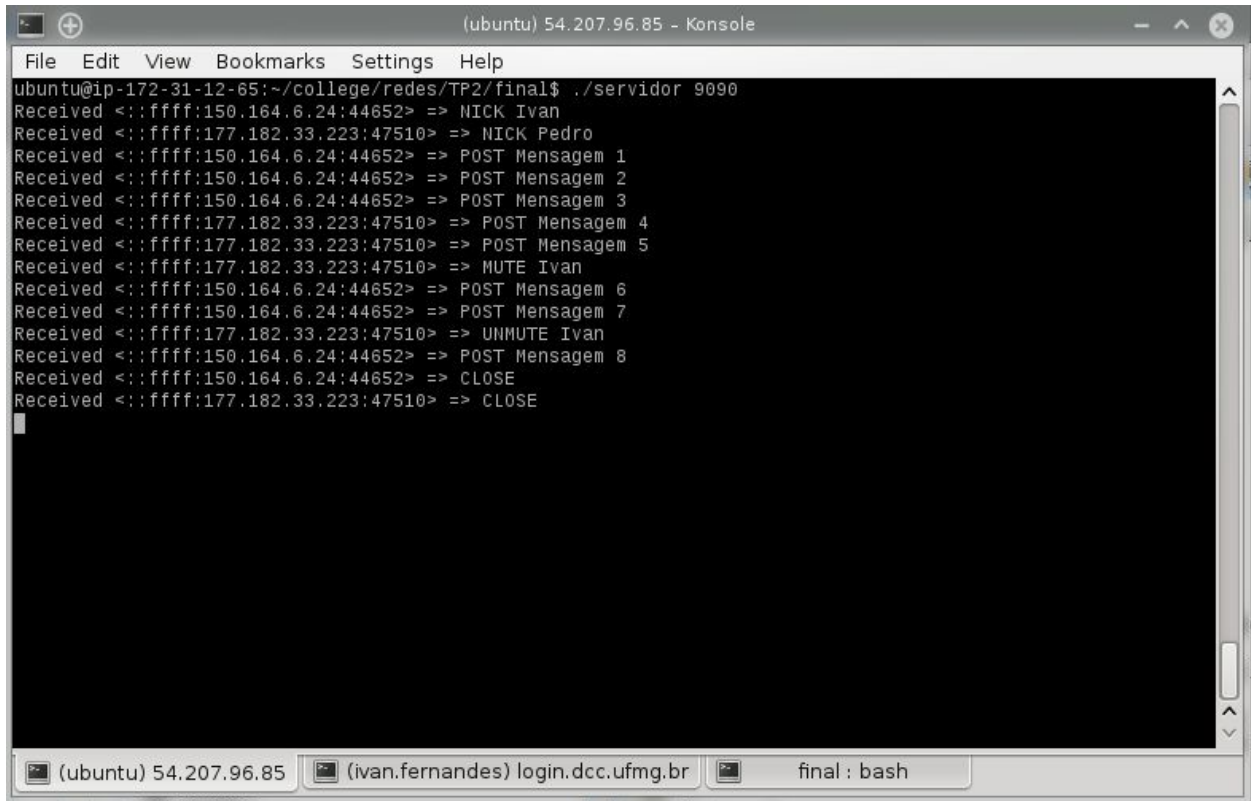
```
(ivan.fernandes) login.dcc.ufmg.br - Konsole
File Edit View Bookmarks Settings Help
claro:~/college/redes/TP2/final> ./cliente doce.grad 9090 Ivan
Username ja cadastrado
claro:~/college/redes/TP2/final> ./cliente doce.grad 9090 Pedro
Conectado como Pedro
Digite a mensagem a ser enviada, ou /help para listar os comandos
msg1
mensagem grande com muitos caracteres
/mute Ivan
O usuário Ivan foi colocado no mudo
/unmute Ivan
O usuário Ivan foi retirado do mudo
Ivan: mensagem fora do mudo
Ivan: mensagem 2 ola
/close
Desconectado com sucesso
claro:~/college/redes/TP2/final> █
```

Cliente 2

Rede WAN

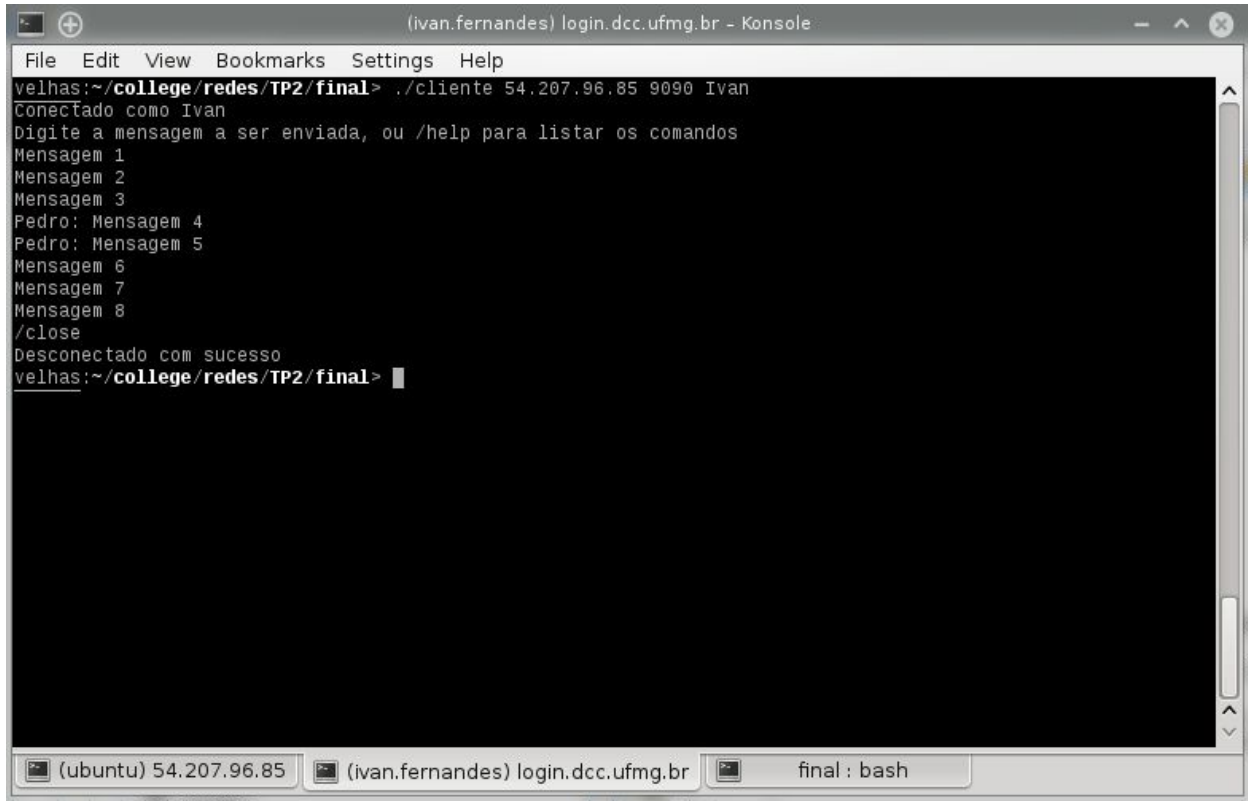
Para o teste na rede WAN, os programas foram executados em 3 redes diferentes. O servidor foi executado em uma instancia EC2 do Amazon Web Services hospedado em São Paulo, um cliente foi executado na maquina velhas.grad do laboratório linux do dcc e o segundo cliente foi executado no minha maquina local.

Como a instancia amazon fornece um endereço de nome public com a instancia, um dos cliente foi conectado usando o ip da maquina enquanto o outro foi conectado pelo nome. O servidor ouvia a porta 9090



```
(ubuntu) 54.207.96.85 - Konsole
File Edit View Bookmarks Settings Help
ubuntu@ip-172-31-12-65:~/college/redes/TP2/final$ ./servidor 9090
Received <::ffff:150.164.6.24:44652> => NICK Ivan
Received <::ffff:177.182.33.223:47510> => NICK Pedro
Received <::ffff:150.164.6.24:44652> => POST Mensagem 1
Received <::ffff:150.164.6.24:44652> => POST Mensagem 2
Received <::ffff:150.164.6.24:44652> => POST Mensagem 3
Received <::ffff:177.182.33.223:47510> => POST Mensagem 4
Received <::ffff:177.182.33.223:47510> => POST Mensagem 5
Received <::ffff:177.182.33.223:47510> => MUTE Ivan
Received <::ffff:150.164.6.24:44652> => POST Mensagem 6
Received <::ffff:150.164.6.24:44652> => POST Mensagem 7
Received <::ffff:177.182.33.223:47510> => UNMUTE Ivan
Received <::ffff:150.164.6.24:44652> => POST Mensagem 8
Received <::ffff:150.164.6.24:44652> => CLOSE
Received <::ffff:177.182.33.223:47510> => CLOSE
```

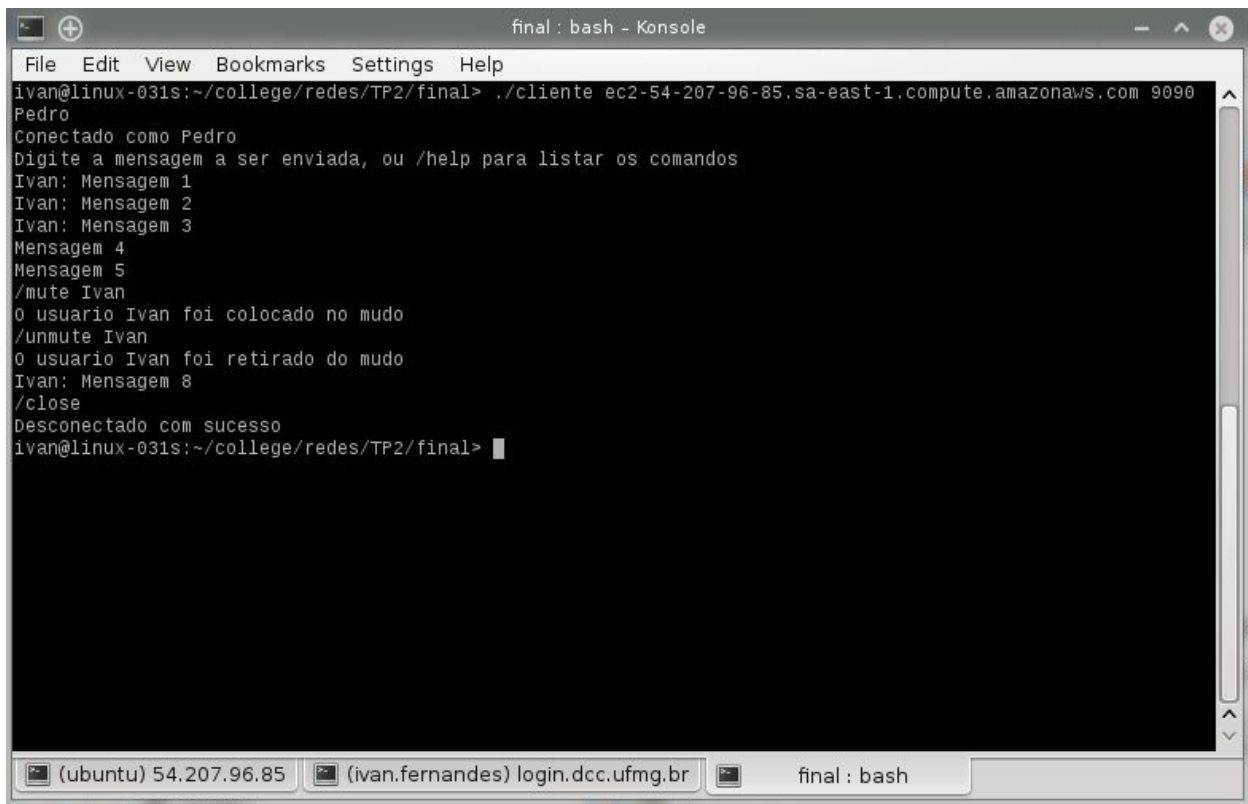
Servidor



A terminal window titled "(ivan.fernandes) login.dcc.ufmg.br - Konsole". The prompt is "velhas:~/college/redes/TP2/final>". The user enters the command ". /cliente 54.207.96.85 9090 Ivan". The output shows the connection process, a list of messages (1-8), and a successful disconnection. The terminal has a menu bar with File, Edit, View, Bookmarks, Settings, and Help. The bottom status bar shows three tabs: "(ubuntu) 54.207.96.85", "(ivan.fernandes) login.dcc.ufmg.br", and "final : bash".

```
File Edit View Bookmarks Settings Help
velhas:~/college/redes/TP2/final> ./cliente 54.207.96.85 9090 Ivan
Conectado como Ivan
Digite a mensagem a ser enviada, ou /help para listar os comandos
Mensagem 1
Mensagem 2
Mensagem 3
Pedro: Mensagem 4
Pedro: Mensagem 5
Mensagem 6
Mensagem 7
Mensagem 8
/close
Desconectado com sucesso
velhas:~/college/redes/TP2/final> 
```

Cliente 1



A terminal window titled "final : bash - Konsole". The prompt is "ivan@linux-031s:~/college/redes/TP2/final>". The user enters the command ". /cliente ec2-54-207-96-85.sa-east-1.compute.amazonaws.com 9090 Pedro". The output shows the connection process, a list of messages (1-8), and a successful disconnection. The terminal has a menu bar with File, Edit, View, Bookmarks, Settings, and Help. The bottom status bar shows three tabs: "(ubuntu) 54.207.96.85", "(ivan.fernandes) login.dcc.ufmg.br", and "final : bash".

```
File Edit View Bookmarks Settings Help
ivan@linux-031s:~/college/redes/TP2/final> ./cliente ec2-54-207-96-85.sa-east-1.compute.amazonaws.com 9090 Pedro
Conectado como Pedro
Digite a mensagem a ser enviada, ou /help para listar os comandos
Ivan: Mensagem 1
Ivan: Mensagem 2
Ivan: Mensagem 3
Mensagem 4
Mensagem 5
/mute Ivan
O usuario Ivan foi colocado no mudo
/unmute Ivan
O usuario Ivan foi retirado do mudo
Ivan: Mensagem 8
/close
Desconectado com sucesso
ivan@linux-031s:~/college/redes/TP2/final> 
```

Cliente 2

Em todos os testes o programas se comportaram como esperado. As mensagens foram todas enviadas e recebidas com sucesso, bem como os comandos do cliente.

Conclusão

O trabalho foi implementado com sucesso, com ambos os programas executando como o esperado em diversos tipos de configurações de rede. O trabalho permitiu exercitar os conceitos vistos em salas sobre os protocolos da camada de transporte, como a confirmação de pacotes, bem como a programação em sockets usando a linguagem C e seus desafios.