WhatsPPD

Trabalho 2 – Programação Paralela e Distribuída

Maiki Buffet e Marcelo Pereira

Estudantes de Graduação em Engenharia de Computação

Pontifícia Universidade Católica do Rio Grande do Sul

Porto Alegre, Brasil

*Resumo*—O trabalho atual apresenta a implementação e funcionalidade de um algoritmo distribuído usando o protocolo TCP-IP de uma simulação de um aplicativo WhatsApp desenvolvido na linguagem de programação C.

Palavras-chaves—Algoritmo Distribuído; TCP-IP; Programação; WhatsApp.

# Introdução

O seguinte trabalho visa apresentar a implementação de um algoritmo distribuído, utilizando o protocolo TCP-IP, para simular o aplicativo WhatsApp, desenvolvido na linguagem C.

O protocolo TCP-IP é uma forma de comunicação entre processos, onde um destes realiza requisições (cliente) a outro que as identifica e as responde (servidor).

As operações que serão abordadas nesta implementação são: adicionar contatos e grupos, enviar mensagens a contatos e grupos, listar as mensagens enviadas e recebidas, além de listar todos os contatos e grupos do usuário.

# Funcionamento

O algoritmo implementado utiliza de duas *threads*, onde uma é responsável pela leitura e identificação dos comandos via terminal, e a outra é responsável pelo recebimento de requisições de ações (servidor).

Nesta implementação os comandos que poderiam ser realizados são:

* Inserção de contatos (#i <nome> <ip>);
* Inserção de grupos (#g <nome1> <nome2> ...);
* Envio de mensagens (#s <nome/grupo> <mensagem>);
* Listagem de mensagens (#l <nome/grupo>);
* Listagem de contato e grupos (#c).

# Instalação

Para a instalação do projeto é necessário apenas estar em posse dos arquivos fontes. A compilação do projeto é realizada de maneira automatizada a partir de um *shell* *script* (compile.sh), que pode ser encontrado junto aos códigos fonte da implementação.

# Implementação

O trabalho foi desenvolvido em ambiente Linux e implementado na linguagem C, utilizando a biblioteca *lpthread* e como compilador o *gcc* versão 4.8.4.

Para a realização deste trabalho foram criadas diversas estruturas de dados (*structs*), que são:

* User: contém o nome do usuário, o caminho do arquivo onde suas informações são armazenadas, o número de contatos, o número de grupos e os vetores de contatos e grupos;
* Contato: contém nome, IP e o caminho para o arquivo onde as mensagens enviadas por este contato são armazenadas;
* Grupo: contém o nome do grupo, número de contatos do grupo, o vetor contendo os contatos do grupo e o caminho do arquivo onde as mensagens deste grupo serão armazenadas;
* Mensagem: contém o tipo de mensagem, o estado em que ela se encontra, o nome do emissor da mensagem e a mensagem propriamente dita;
* ClientInfo: contém o endereço de IP para onde será enviada a mensagem e um ponteiro para a mensagem que será enviada.

O primeiro passo da execução é realizar o login do usuário, isto é realizado a partir da leitura de um nome via terminal, é então verificada a existência de um usuário com este nome e caso não exista ele é criado.

Caso o usuário, com o nome identificado no método de login, já exista no diretório *User* do projeto, suas informações são carregadas a partir de um arquivo binário de mesmo nome do usuário e extensão ‘u’.

Após isto, o programa começa a esperar por comandos. As entradas dadas pelo terminal são identificadas utilizando um *parser*, este que identificará comandos válidos ou não. Se um comando válido for identificado, é chamada uma rotina de tratamento para determinado comando.

As rotinas de tratamento de comandos criadas foram:

* executeCommandInsertion:

Utilizando um *parser*, é identificado o nome do contato a ser adicionado e seu endereço de IP, após isto, verifica-se se este contato já existe na lista de contatos do usuário corrente.

Caso não exista um contato com o nome ou IP informados, é enviada uma mensagem ao contato solicitando a inserção do usuário que fez a requisição, caso o contato consiga adicionar o usuário em sua lista de contatos, é enviada uma mensagem de retorno confirmando a inserção, só então que o usuário que fez a requisição de inserção de contato poderá adicionar o contato em sua lista.

Se o contato não conseguir adicionar o usuário, é enviada uma mensagem de retorno informando a falha, assim o usuário que fez a requisição de inserção, receberá uma notificação de que o contato não pôde ser adicionado por alguma razão. O motivo do erro vem no campo ‘msg’ da mensagem de retorno de falha, enviada pelo contato.

* executeCommandInsertionGroup:

Mesmo princípio do método executeCommandInsertion, com a única diferença que faz a adição do grupo específico aos contatos do usuário.

* executeCommandSend:

Utilizando um *parser*, identifica-se o nome do contato a quem se deve enviar a mensagem e a mensagem em si, após isto, verifica-se se este contato já existe na lista de contatos do usuário corrente.

Caso o contato exista, a mensagem poderá então ser enviada ao contato – este por sinal quando receber a mensagem – a escreve no arquivo de mensagens correspondente ao nome do usuário que a enviou. Dessa forma, uma mensagem de confirmação de recebimento ao usuário retornará, só então o usuário que enviou a mensagem original, poderá escrevê-la no arquivo de mensagens daquele contato.

* executeCommandList:

Utilizando um *parser*, identifica-se o nome de qual contato serão listadas as mensagens, após isto, é realizada a verificação se este contato existe na lista de contatos do usuário corrente.

Caso o contato exista, procura-se pelo arquivo de mensagens que tem o mesmo nome que o contato, então se abre o arquivo e se mostra o conteúdo dele no terminal. Quando é detectado o fim do arquivo, o cursor é levado novamente ao início do mesmo para que seja adicionado um ‘\*’ ao final das mensagens recebidas, a fim de sinalizar que estas foram lidas.

Depois de realizar todas as modificações nas mensagens recebidas, é enviada ao contato uma mensagem de aviso de leitura, assim o contato pode atualizar o seu arquivo de mensagens, colocando o segundo ‘\*’ no final das mensagens enviadas por ele.

* printUserContacts:

Este método apenas itera sobre as listas de contatos e grupos de um usuário, imprimindo os nomes dos tais.

Todos os envios de mensagens do programa utilizam o mesmo método, chamado initClient. Este método é responsável por abrir e conectar os *sockets* de comunicação com os servidores dos contatos, enviar uma mensagem pelo *socket* criado, e após, esperar por uma mensagem de confirmação.

Nesta implementação existem dez tipos de mensagens e cada um define o modo como o servidor deverá processá-las. Os tipos de mensagens existentes são:

* MSG\_TXT: indica que é uma mensagem de texto;
* MSG\_TXT\_GROUP: indica que é uma mensagem de texto de grupo;
* ADD\_CONTACT: indica que se deve adicionar o usuário que enviou esta mensagem na lista de contatos do usuário que recebeu;
* ADD\_GROUP: indica que se deve adicionar o grupo e os integrantes do mesmo e o criador - que enviou esta mensagem - na lista de contatos dos usuários que receberam o convite;
* READ\_MSGS: indica que deve ser atualizado o arquivo de mensagens de um contato;
* READ\_MSGS\_GROUP: indica que deve ser atualizado o arquivo de mensagens de um grupo;
* RET\_MSG\_OK: indica que a mensagem enviada anteriormente foi devidamente processada;
* RET\_MSG\_FAIL: indica que a mensagem enviada anteriormente não foi devidamente processada;
* RET\_MSG\_OK\_GROUP: indica que a mensagem enviada anteriormente foi devidamente processada pelo grupo;
* RET\_MSG\_FAIL\_GROUP: indica que a mensagem enviada anteriormente não foi devidamente processada pelo grupo.

O servidor do projeto é inicializado em uma *thread* logo após o *login* do usuário, assim o programa está apto a receber as mensagens de outros clientes.

O método do servidor é responsável por criar um *socket*, associá-lo a uma porta da máquina, e ficar em espera por conexões.

Quando um cliente realizar uma conexão com o servidor, ele cria um novo *socket* para comunicação com o cliente específico, só então o servidor está pronto para receber as mensagens do cliente que está conectado.

Após recebimento de uma mensagem, o servidor irá identificar o seu tipo, para então tratá-la devidamente.

No caso de mensagens de texto, o servidor irá verificar a existência do contato que enviou a mensagem na lista de contatos do usuário, para então escrevê-la em um arquivo, com o nome do emissor da mensagem como título e de extensão ‘msg’. Após a mensagem ser devidamente armazenada, o servidor cria e envia uma mensagem de retorno confirmando o seu recebimento. Se for o caso de mensagens para grupo, esta mensagem é replicada para todos os contatos integrantes do mesmo, seguindo o mesmo procedimento descrito anteriormente.

No caso de inserção de contato, o servidor irá verificar a existência do contato que enviou a mensagens na lista de contatos do usuário e depois verificará se o usuário tem espaço disponível para mais um contato, só então o novo contato será adicionado. Após isso o servidor enviará uma mensagem de confirmação da execução de inserção. Se for o caso de inserção de grupo, este pedido é replicado para todos os contatos integrantes do mesmo, seguindo o mesmo procedimento descrito anteriormente.

No caso de mensagem de listagem, o servidor irá verificar a existência do contato que enviou a mensagem na lista de contatos do usuário, para então reescrever o arquivo de mensagens correspondente ao emissor da mensagem, colocando o sinalizador ‘\*’, indicando que as mensagens enviadas pelo usuário foram lidas pelo contato emissor. Se for o caso de mensagem de listagem de grupo, este pedido é replicado para todos os contatos integrantes do mesmo, seguindo o mesmo procedimento descrito anteriormente.

A atual implementação não suporta a checagem de status se um contato está *online* ou não, sendo necessário que todos contatos envolvidos estejam logados. Em função disto, mensagens enviadas para contatos *offline* ou adições em grupos não funcionam.

# Futuros Trabalhos

Para avançar ainda mais no aprendizado deste experimento, planeja-se fazer a criação de um thread paralela que será a responsável por fazer *polling* dos usuários – a fim de verificar se estão *online* ou não. Além disto, um buffer deverá ser implementado para guardar as mensagens não enviadas aos usuários deslogados, para serem enviadas assim que os mesmos estiverem online.

Outros trabalhos futuros planejados são: utilização do nome do usuário que leu a mensagem – no caso dos grupos, além da inclusão da hora da leitura/envio. Adicionar a opção de edição de grupos, podendo-se adicionar ou remover membros, além do fato de poder adicionar membros sem que seja necessário possuir o mesmo em sua lista de contatos.

Para consultas futuras, a ideia é a implementação mais próxima ao modelo atual utilizado por mensageiros como WhatsApp, Messenger e Telegram.