DOSAPP - Relatório do desenvolvimento de uma aplicação de comunicação virtual

Giuliano Barbosa Prado[1], Marcello de Paula Ferreira Costa[1]

[1]Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo (USP)
Escola de Engenharia de São Carlos – Universidade de São Paulo (USP)
São Carlos – SP – Brazil

giuliano.prado@usp.br, marcello.costa@usp.br

Professor Responsável: Júlio Cezar Estrella[2]

[1]Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo (USP)
São Carlos – SP – Brazil

jcezar@icmc.usp.br

Abstract. *This paper describes the operation and development of a virtual communication application. It describes the methods to communicate, as well as techniques to control active connections and recover from failures. The system is developed under a pure Peer-to-Peer architecture, designed in a mesh network that runs over TCP/IP.*

Resumo. *Este artigo descreve o funcionamento e desenvolvimento de uma aplicação de comunicação virtual. Apresenta os métodos utilizados para efetuar a comunicação, bem como técnicas para manter conexões ativas e recuperar de falhas. O sistema desenvolvido conta com arquitetura Peer-To-Peer pura, projetado em uma rede em malha que executa sobre o protocolo TCP/IP.*

1. Introdução

Métodos de comunicação são utilizados por seres vivos desde os primórdios da humanidade. Nos períodos pré-históricos, linguagens rupestres e técnicas arcaicas de sinais foram devidamente desenvolvidas para permitir que seres humanos se comunicassem e conseguissem transmitir suas intenções e planos.

Nessa evolução, a capacidade de fala foi aprimorada e línguas arcaicas foram criadas, facilitando a transmissão de conhecimento e planejamento estratégico das civilizações. Juntamente a essa evolução, novos meios de comunicação foram sendo criados e evoluídos, surgindo, por exemplo, sistemas de correios.

Com o advento da computação, meios de comunicação digitais tornaram-se necessários. Desenvolver uma rede de internet tornou-se essencial para prover um método de comunicação de abrangência global e com boa eficiência. Esse trabalho focou-se na criação de um sistema de comunicação digital, visto que o desenvolvimento de um sistema desse tipo envolve um conhecimento teórico abrangente e oferece a possibilidade de aprender como programar aplicações voltadas a web, complementando o aprendizado teórico apresentado pelas disciplinas de rede.

2. Descrição do problema

O problema cuja resolução é proposta pelo projeto consiste em um sistema peer-to-peer de comunicação virtual (sistema de troca de mensagens), no qual usuários podem enviar mensagens uns aos outros, ou para múltiplos contatos.

Esse sistema deverá oferecer ao usuário funcionalidades similares a um mensageiro virtual ou email, permitindo que os mesmos criem suas listas de contatos e adicionem os nós com que desejam conversar. Ele deve conter um método de checagem de mensagens recebidas, adicionar, listar e excluir contatos, além do envio de mensagens em grupo e individuais. Todas essas funcionalidades devem ser apresentadas de forma clara e simples ao usuário, permitindo uma utilização rápida e prática.

Além disso, é importante que o aplicativo contenha um sistema transparente para fechamento de conexões, permitindo ao usuário sair do processo sem que isso prejudique os seus contatos, além de que esses controle deva contemplar usuários cuja conexão caia durante o processo, removendo estes da lista de contato dos seus pares.

3. Arquiteturas e protocolos adotados

3.1. Protocolo TCP/IP

O protocolo IP (Internet Protocol) é o mais utilizado na atualidade, sendo aquele que suporta praticamente todos os serviços vistos nas redes domésticas. Nesse contexto, ele se torna o mais adequado a aplicação discutida nesse relatório devido a sua grande utilização, facilidade de uso e disponibilidade de bibliotecas prontas para trabalhar.

Já o protocolo TCP (Transmission Control Protocol) provê uma versatilidade e robustez adequada a aplicação proposta, já que este verifica se os dados são enviados da forma correta, na sequência correta e sem erros, pela rede. Assim, como ele oferece uma transmissão livre de erros e com garantia de entrega, ele é de grande utilidade para nosso projeto, já que o foco do projeto é o envio de mensagens e essas não devem ser perdidas ou ter ordem mudadas. A fim de comparação, o protocolo UDP não é confiável e não garante se o pacote irá chegar ou não. Assim, utilizar o UDP poderia tornar o sistema suscetível a perda de pacotes ou requerer um controle maior para evitar esses fatos.

3.2. Arquitetura Peer-To-Peer (P2P)

O peer-to-peer é uma arquitetura onde cada nó funciona tanto como cliente quanto como servidor. Assim, cada nó conecta diretamente ao outro sem a necessidade de um servidor central. Isso traz como grande vantagem a independência da comunicação entre as unidades, o que dá mais segurança, já que um servidor não tem a informação de todas as comunicações, além da estabilidade, já que a queda de um nó não derruba o sistema todo.

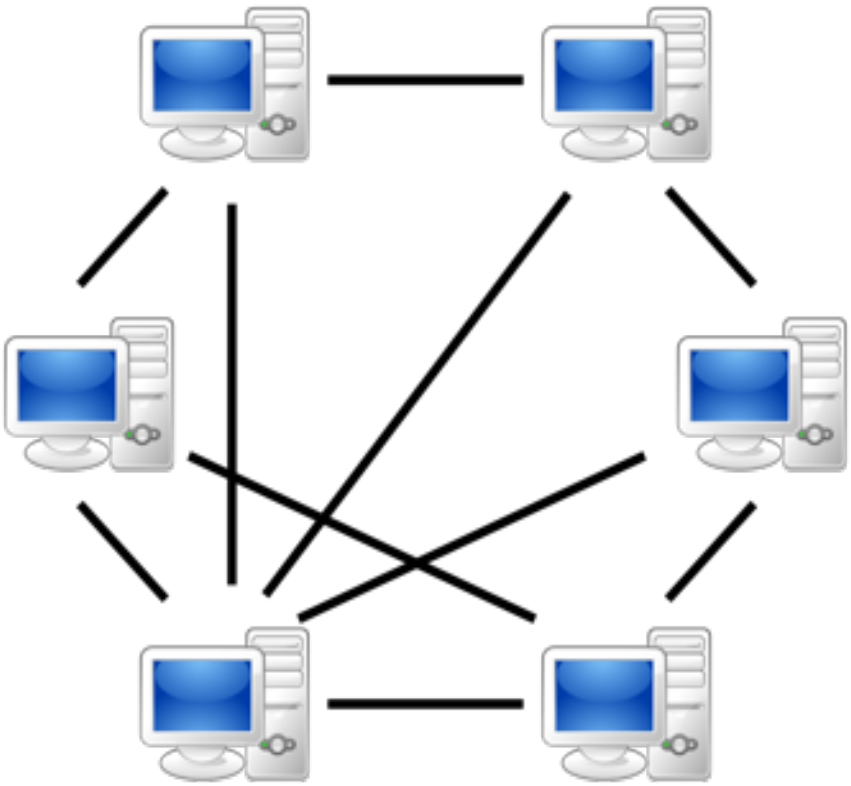

Figura 1. Rede Peer-To-Peer

Nesse caso, o modelo adotado no projeto consiste, resumidamente, em um processo que contém uma thread servidora, para receber conexões, e threads que solicitam conexão, formando um sistema peer-to-peer puro.

3.3. Rede Mesh Parcial

A estrutura escolhida para o projeto é em malha (mesh), sendo que cada nó conecta com o nó que deseja comunicar, garantindo assim uma conexão direta entre eles. Assim, não são necessários intermediários para a comunicação, oferecendo mais segurança e rapidez no processo.

Um ponto importante a ressaltar é que a rede não é puramente em malha. Assim, não é necessário que os nós se comuniquem com todos os outros, já que um nó só necessita conectar à aquele que deseja enviar mensagem, ou receber conexão daquele que deseja falar com ele. Desse modo, as conexões não deverão sobrecarregar a rede, já que só são criadas aquelas para cada conversa, o que, de modo geral, não são muitas.

4. Funcionamento

O modelo adotado para resolução do problema proposto buscou encapsular ao máximo as funcionalidades e estruturas para controle da rede. Essa metodologia foi adotada visando tornar o código mais claro para análise, já que deixa visível, através de nomes significativos e explicações, qual a intenção daquele fragmento e como seu uso deve ser feito. Além disso, torna-se mais fácil e claro o gerenciamento da rede, pois as funções criadas encapsulam o conceito e deixam mais prática a sua utilização.

Vale ressaltar também que o projeto foi dividido em 6 arquivos: contacts.c, contacts.h, dosapp.c, lib.c, messages.c e messages.h

Nessa seção desse documento, é explicado o conteúdo de cada um dos arquivos, além de demonstrar o funcionamento básico de cada uma das funções desenvolvidas e utilizadas no projeto.

4.1. O arquivo lib.c

O arquivo lib.c contém as principais funcionalidades de rede do sistema. Abaixo será descrito o funcionamento e o contexto de cada função.

4.1.1. Inicialização do servidor

A função *criarSocketServidor* encapsula todas as chamadas necessárias para se criar um socket de servidor.

Inicialmente, é criado um socket utilizando como parâmetros AF_INET, que indica que é um socket internet, e SOCK_STREAM, que provê uma conexão confiável, de mão dupla e baseada em conexão (TCP).

Após a criação, são definidos os atributos do endereço do servidor. Em seguida, é definida a opção SO_REUSEADDR, que permite que um socket criado aceite múltiplas requisições. É necessário, então, fazer um *bind* no socket, informando ao mesmo o endereço.

Por fim, utilizamos a função *listen*, que faz com que o socket comece a escutar naquele IP, na sua devida porta.

Essa função é chamada no escopo principal e logo em seguida uma thread responsável por aceitar conexões na variável global sockServer é criada. A variável sockServer só será de uso válido caso a função *criarSocketServidor* completar com sucesso, e retornar, portanto, o socket criado.

4.1.2. Solicitando uma conexão

Para que seja criada uma nova conexão, primeiramente um servidor deve estar ativo no lado destinatário e os argumentos *serverName* e *portNumber* devem ser corretos. A função *solicitarConexao* recebe justamente os dois parâmetros mencionados e realiza a chamada da função *connect,* que colocará um valor de socket válido no primeiro argumento passado à ela.

A função *solicitarConexao* é chamada no escopo principal e é usada como condição para que um contato seja adicionado à lista de contatos. Assim, essa função é utilizada nas criações e o socket retornado é colocado no campo correspondente dentro de uma *struct* do tipo contact.

É importante ressaltar que dentro da função para solicitar uma conexão, uma rotina que estipula um *timeout* é realizada. Essa funcionalidade permite que, caso não sejam recebidos novos pacotes, a função de *receive* (que, na nossa aplicação, é bloqueante) desbloqueie e defina um erro na variável do sistema *errno*, permitindo efetuar seu tratamento. [detalhes no item 5.1.]

4.1.3. Aceitando uma conexão

A função responsável por aceitar uma conexão é chamada *recebeConexao*, esta que deve receber um parâmetro do tipo *Socket* (inteiro) para saber qual socket deve usar na chamada da função *accept*. A função accept bloqueia a thread que a executa e quando uma solicitação é recebida, desbloqueia e retorna um novo socket para essa nova conexão TCP criada. A função *recebeConexao* realiza também uma rotina para estipular um *timeout*, que é tem valor igual usado na função *solicitarConexao*. O novo socket com o recurso de *timeout* é então retornado.

4.1.4. Enviando e Recebendo dados

Primeiramente é necessário dizer que um protocolo bastante simples apelidado de DAPP foi definido na camada de aplicação. A mensagem da camada de aplicação é simplesmente um caractere numérico indicando qual o tipo da mensagem (Mensagem de dados, Mensagem Ack e Mensagem de atualização de dados), seguido de <>, seguido da mensagem efetiva. Em outras palavras, o cabeçalho é composto somente por um caractere numérico seguido de <> para indicar fim de cabeçalho.

A função *enviarDados*, responsável por fazer o envio da mensagem de tipo correto, recebe como argumento um valor inteiro que a ajuda decidir qual o tipo de mensagem enviará. O tipo será o primeiro caractere da mensagem, seguido de <>, seguido da mensagem, como definido pelo protocolo DAPP.

A função *receberDados* não faz distinção entre os tipos de mensagem. O texto é somente lido e armazenado em um *buffer* dinâmico, que é tratado em nível superior para distinguir entre os três tipos de mensagem disponíveis.

4.2. Os arquivos contacts.c e contacts.h

O arquivo de cabeçalho (contacts.h) contém as definições de tipos e inclusão das bibliotecas usadas para a implementação do código contido no arquivo contacts.c.

No arquivo cabeçalho são definidas duas estruturas que implementam uma lista duplamente encadeada. Uma delas define o formato de cada elemento dessa lista e é definida com o nome *contact*, a outra contém informações relevantes para iteração e controle sobre a lista e é chamada de infoList.

A descrição detalhada das funcionalidades desses arquivos não cabem ao escopo deste relatório, que tem o intuito de descreve os conceitos de Redes de Computadores utilizados.

4.2.1. Funcionalidades da Lista de Contatos

A implementação do TAD de lista duplamente encadeada conta com as funções de inserção e remoção de contatos. Além disso, estão disponíveis funções de busca por uma chave (String) e impressão em tela de todos os contatos. Por fim, a função de inicialização, que inicia uma lista vazia, e a função de esvaziamento, que remove todos os contatos da lista, mas não a destrói por completo.

4.3. Os arquivos messages.c e messages.h

O arquivo de cabeçalho (messages.h) contém as definições de tipos e inclusão das bibliotecas usadas para a implementação do código contido no arquivo messages.c.

No arquivo cabeçalho são definidas duas estruturas que implementam uma lista encadeada. Uma delas define o formato de cada elemento dessa lista e é definida com o nome message, a outra contém informações relevantes para iteração e controle sobre a lista e é chamada de messageList.

A descrição detalhada das funcionalidades desses arquivos não cabem ao escopo deste relatório, que tem o intuito de descreve os conceitos de Redes de Computadores utilizados.

4.3.1. Funcionalidades da Lista de Mensagens

A implementação do TAD de lista encadeada conta com as funções de inserção e remoção de mensagens. Além disso, a função de impressão em tela de todos os contatos e por fim, a função de inicialização, que inicia uma lista vazia, e a função de esvaziamento, que remove todas as mensagens da lista, mas não a destrói por completo.

4.4. O arquivo dosapp.c

O arquivo dosapp.c contém o núcleo do sistema, bem como o controle das threads e da main. A função main contém o controle principal do menu, bem como a chamada a outras funções, além da criação das threads básicas. Funções auxiliares, como a de leitura para o menu e leitura dos dados digitados pelo usuário estão disponíveis no arquivo

Além disso, esse arquivo contém todas as threads que serão executadas no programa. A thread leitora é criada para cada uma das conexões e é responsável por receber novas mensagens, sejam elas ACKs, nomes ou texto. Já a thread aceitadora é responsável por executar em background recebendo novas conexões para aquele nó, já que todos eles são clientes e servidores ao mesmo tempo. A thread ACK, por outro lado, fica responsável por enviar mensagens de ACK para os outros nós, informando que aquele processo continua ativo. Por fim, temos a thread eraser, que limpa nós inválidos no programa.

5. Queda de conexão

Um fator relevante a ser considerado no projeto é a estabilidade do aplicativo. É necessário que a queda de um nó não afete o sistema inteiro. Assim, foram tomadas estratégias buscando evitar esse tipo de programa e permitindo ao sistema perceber esses acontecimentos de forma transparente ao usuário.

5.1. Sistema de *timeout*

As primitivas *send* e *receive* utilizadas no desenvolvimento do projeto foram utilizados em sua forma bloqueante. Isso indica que a thread que as chamam fica bloqueada até o recebimento/ envio de uma mensagem. Assim, utilizamos uma thread leitora, por exemplo, que chamará constantemente a função *receive* para cada thread, fazendo com que armazene as mensagens recebidas. Vale ressaltar que essa função é bloqueante: logo, a thread fica aguardando a resposta dessa função.

Nesse contexto, torna-se importante efetuar um desbloqueio quando é removida uma thread, evitando problemas no sistema. Dessa forma, foi definida a opção *SO_RCVTIMEO* em cada socket, o que faz com que os bloqueios de rede naquele socket possuam um *timeout* de um tempo definido. No fim desse tempo de *timeout*, caso não sejam recebidos dados, a variável do sistema *errno* é definida com um código de erro, sendo possível remover esse nó nesse caso. Para que esse sistema funcione corretamente, é fundamental um sistema de ACKs, que garantirá que o nó não cairá por *timeout* durante a execução normal.

5.2. Sistema de ACKs

ACK ou Acknowledge é um pacote enviado periodicamente por um nó para sinalizar sua presença aos contatos que estão conectados à ele. Assim, a cada ciclo de tempo, todos os nós ligados ao remetente são informados do estado da conexão, como é possível observar na figura 2.

Para evitar que a conexão seja desfeita durante uma execução convencional, foi criada uma thread responsável por enviar ACKs a todos os nós conectados a cada 3 segundos. Isso garante que, caso a conexão esteja em curso, ela não será descartada por *timeout*, já que haverão novos pacotes chegando constantemente na função *receive*, mantendo o sistema ativo.

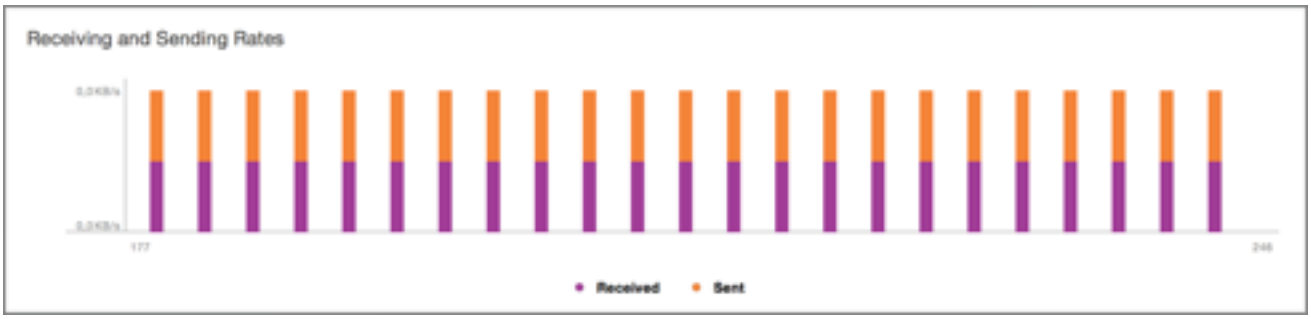

Figura 2 - Sinais de ACK sendo enviados e recebidos

5.3. Funcionamento do sistema de queda de conexão

Para evitar condições de corrida, existe uma thread separa responsável por apagar os contatos inválidos. A ideia é que se algum problema de escrita ou leitura em conexões inexistentes ou um *timeout* seja verificado, a thread que verificou o acontecimento marca o contato como inválido.

Esse contato inválido é removido pela thread *threadEraser*. Para evitar maiores problemas, o loop de envio de Acks e a ação de remover um contato estão envolvidas por um mutex, evitando quaisquer problemas de segmentação.

O usuário não verá o contato inválido em sua lista de contatos mesmo que a thread responsável pela limpeza ainda não o tenha removido por completo da lista. Também não é possível enviar uma mensagem para esse contato inválido, já que esse parâmetro é sempre verificado antes de qualquer envio de mensagem.

6. Menu principal, funcionalidades e acontecimentos em tempo de execução

Figura 3 - Menu principal

O menu principal é exibido logo após a escolha do seu nome. É importante dizer que ao iniciar o programa já inicia a thread que aceitará conexões, ou seja, a parte *servidor* do DosApp já está em funcionamento antes mesmo de você escolher o seu nome.

Qualquer conexão recebida por essa thread, acarretará em um novo contato criado adicionado em sua lista de contatos e uma thread dedicada à receber as mensagens desta conexão. Esta que terminará sua execução por conta própria caso qualquer problema na rede seja identificado.

6.1. Inserir contato

Essa opção pergunta ao usuário o nome que será dado ao novo contato e o endereço IP deste novo contato. A adição do contato à lista só ocorrerá se a conexão puder ser estabelecida com sucesso. Não podem haver duplicatas de nomes e também a palavra “quit” é reservada para uso do processo.

6.2. Listar contatos

Exibe na tela a lista de contatos válidos. Contatos inválidos possivelmente estão na iminência de serem excluídos.

6.3. Excluir contato

Requisita ao usuário um nome de contato para ser excluído. Somente será executada uma exclusão caso o nome seja encontrado em algum contato da lista de contatos.

6.4. Enviar mensagem para um contatos

Primeiramente é pedido ao usuário que digite uma mensagem. Esta que será o texto enviado ao destinatário, requisitado posteriormente ao usuário. Óbvio que um envio ocorrerá se, e somente se existir um contato com o nome fornecido.

6.5. Enviar mensagem para todos os contatos

É requisitado ao usuário que digite uma mensagem. Esta que será o texto enviado para todos os contatos em sua lista de contatos.

6.6. Enviar mensagem para contatos selecionados

Nesta opção, o processo pede uma mensagem e em seguida mostra a lista de contatos. À partir deste momento, o usuário tem a opção de enviar para quais contatos ele quiser. A seleção é feita via requisição de nome e busca como na opção 6.4. Vale lembrar que essa opção não filtra o caso de envio repetido, ou seja, se o usuário quiser, pode mandar a mensagem para o mesmo usuário várias vezes. O número de contatos é o número de envios possíveis. Para sair desta opção basta digitar "quit".

6.7. Listar mensagens recebidas

Um nome é pedido e então, se este nome constar na lista de contatos, as mensagens recebidas à partir da última visualização são mostradas em tela. Essas mensagens são imediatamente apagadas da caixa de entrada.

6.8. Sair

Inicia rotina de finalização e sai do loop do menu principal. Isso implica em uma posterior marcação de flags para que as threads servidor e ack possam finalizar seu trabalho.

7. Conclusão

Esse projeto permitiu o melhor entendimento de redes de computadores, bem como o desenvolvimento de aplicações web. Foi possível perceber os erros mais comuns inerentes a esse tipo de sistema, além de estratégias para contornar os mesmos. Foi possível também aprimorar o aprendizado em outras áreas inter-relacionadas, como programação utilizando threads, algo que se torna indispensável para uma aplicação desse nível em C.

Por fim, foi possível aplicar os conhecimentos teóricos das disciplinas de redes em um projeto real, dando um enfoque prático as utilidades do conteúdo estudado e permitindo um amadurecimento no conteúdo

Referencias

Kurose & Ross, Redes de Computadores e a Internet, 3a ed., 2006

Tanenbaum, Redes de Computadores, 4a ed., 2003

Estrella, Júlio, Notas de Aula e exemplos práticos de implementação de conexões.

Wikipedia, Transmission Control Protocol. Disponível em <https://pt.wikipedia.org/wiki/Transmission_Control_Protocol>. Acesso em 28 de junho de 2015.

Wikipedia, Peer-to-Peer. Disponível em <https://pt.wikipedia.org/wiki/Peer-to-peer>. Acesso em 28 de junho de 2015.

Wikipedia, Redes Mesh. Disponível em <https://pt.wikipedia.org/wiki/Redes_Mesh>. Acesso em 28 de junho de 2015.

Wikipedia, Protocolo IP. Disponível em <https://pt.wikipedia.org/wiki/Protocolo_de_Internet>. Acesso em 28 de junho de 2015.