DIG-DCC023 - REDES DE COMPUTADORES

TRABALHO PRÁTICO 1

Aluno: Danilo Pimentel de Carvalho Costa
Matrícula 2016058077

Introdução

O Trabalho Prático 1 da disciplina de Redes de Computadores visa aplicar os conceitos aprendidos em um sistema de subscrição e publicação de tweets. O programa deve usar biblioteca padrão com interface POSIX, para ter bom entendimento do processo de comunicação em rede sem mais abstrações.

Servidor e cliente básicos

O primeiro desafio percebido foi a implementação da comunicação básica entre servidor e cliente(s). Foi utilizado o código do professor Ítalo para começar o projeto, o que facilitou o começo do sistema. Com este ponto de partida, foi possível estabelecer a conexão e enviar mensagens.

A especificação do trabalho estabelece as diretrizes básicas de comunicação entre o cliente e servidor, como o trabalho da mensagem (500 bytes), proibição de transmissão de caracteres '\0' na rede, fim de mensagem em '\n'. Houve dificuldade de percepção do caractere '\0' por não ser visível no terminal ao imprimir mensagens, mas foi facilmente resolvido após alterar o tamanho dos buffers e definir a quantidade certa de caracteres enviados na rede.

A primeira versão do sistema permitia o envio de mensagem de um cliente para todos os clientes. O servidor cria uma thread por cliente, e estas threads precisam da informação de pelo menos o file descriptor do socket de todos os clientes. Foi criada uma estrutura para as informações relevantes a todo o servidor, e dentro desta estrutura é mantido um vetor com as informações de todos os clientes. A estrutura que guarda informações da thread tem um ponteiro para este, e assim todas as threads podem ter acesso aos file descriptors dos sockets de todos os clientes. Desta forma, foi possível o broadcast de mensagens para todos os clientes.

A definição de um protocolo estabelece regras para mensagens válidas. A especificação permite ao servidor desconectar qualquer cliente que viole tais regras. A desconexão de clientes implica na alteração de uma região de memória compartilhada entre threads. Condições de corrida e resultados inesperados podem acontecer durante o envio de mensagens. Na estrutura para gerência de clientes conectados, foi adicionado um lock, e nas operações de conexão, desconexão, envio de mensagens, e parada do servidor este lock foi utilizado para evitar tais problemas de acesso concorrente.

Na validação de caracteres da mensagem, a solução mais natural durante o desenvolvimento foi a utilização de expressões regulares. Este recurso é de fácil utilização na biblioteca padrão do C++. Neste momento, foi decidido a transição para esta linguagem, que posteriormente facilitou a manipulação e utilização de vetores para outras funções do programa.

Cadastrando e descadastrando interesse
Até este momento, as funções de subscrição em tópicos não estavam implementadas. Na estrutura para cada cliente, foi adicionado um vetor que contém tags que o cliente está interessado. Um novo processamento de mensagens recebidas pelo cliente foi adicionado, e caso a mensagem seja do tipo +tag ou - tag, o vetor de tags é manipulado. Com o vetor de tags, bastou alterar o processamento de broadcast de mensagens para checar se o cliente está interessado ou não.

O protocolo estabelece o formato que uma tag deve respeitar, tanto nas operações de definição de interesse, quanto nas mensagens enviadas pelo cliente. A mesma solução de expressões regulares foi aplicada para checar a validade da tag.

Assim como na implementação do servidor básico, existem problemas no acesso à memória compartilhada entre threads. Novamente, para resolver tais problemas foi adicionado um lock na estrutura de cada cliente, para controlar o acesso ao vetor de tags. Nas operações de adição de tag, remoção de tag, e broadcast de mensagem, este lock foi utilizado.

Após a implementação destas funções, o programa atende os requisitos básicos da especificação.

Testes
Foram disponibilizados scripts para a execução de testes automatizados e um teste básico. Após uma exploração, foi descoberto o funcionamento e com isso outros casos de teste puderam ser adicionados. A execução de testes automatizados facilitou a checagem de regressão em alterações posteriores e foi de grande ajuda.

Além disso, a execução do primeiro teste mostrou um requisito que antes não ficou claro na especificação. Se um determinado cliente subscrito no tópico #dota envia uma mensagem contendo esta tag, o servidor também deve enviar esta mensagem de volta para o mesmo cliente. O problema foi identificado e corrigido, e o teste disponibilizado passou.

Pontos a melhorar
O Makefile do projeto contém rotinas para execução do Valgrind. Não foi priorizada a correção de todos os vazamentos de memória. Vazamentos relativos à alocação de memória para as estruturas do programa como dados do servidor e cliente foram resolvidos, bem como a liberação dos file descriptors dos sockets. Porém, isto não foi suficiente. Suspeito que as operações para abertura de conexão e criação de threads fazem outras alocações que o programador deveria liberar, mas não consegui identificá-las antes da finalização do trabalho.

O programa utiliza a linguagem C++, mas começou utilizando a linguagem C. Durante a transição não foi feita uma transformação do código procedural em código orientado a objetos. Vejo que este recurso facilitaria o entendimento e operação do programa, porém não foi priorizado para a entrega do trabalho.

O programa utiliza estratégias ingênuas em alguns pontos, aumentando a complexidade de tempo e espaço em sua execução. Neste trabalho, porém, foi priorizada a facilidade de

entendimento da operação, e portanto a simplicidade da implementação. A otimização destes procedimentos fica pendente e também é um ponto a melhorar.

Conclusão

O trabalho permitiu o entendimento dos desafios na implementação de programas que se comunicam em rede. Pode ser estudado o paradigma de comunicação servidor-cliente, o estabelecimento de um protocolo, programação concorrente, e compartilhamento de memória entre processos. O padrão pub/sub foi um tema interessante para ser trabalhado, pois foi complexo o suficiente para aprender tais tópicos, e simples o suficiente para ser implementado no tempo disponível.