Relatório Trabalho 3

Thiago Bell 228527

7 de Novembro de 2016

1 Ambiente de Desenvolvimento e Teste

A principal plataforma de desenvolvimento foi em uma máquina Linux 3.16.0-4, Debian x86_64, Intel(R) Core(TM) i5-2540M, 6 GiB de Memória e usando GCC 4.9.2. Além dessa, o servidor também foi testado em uma SoC ARMv7 com Linux 3.4.108 e Debian(Bananian). O compilador foi o GCC com a mesma versão da outra máquina.

2 Servidor

O servidor armazena todos os dados referentes aos clientes conectados e as salas existentes. Ele é um servidor multithreaded e, por isso, deve adotar técnicas de sincronização para garantir a integridade das estruturas de dados.

2.1 Arquitetura do Servidor

No servidor, cada cliente é representado com uma thread e uma respectiva estrutura de dados. Essa estrutura armazena o nome de usuário, a sala a qual o usuário pertence, o descritor de socket da conexão com esse respectivo cliente, um Mutex para controle de acesso a escrita no socket e uma variável pthread_t. Cada uma dessas threads é responsável por esperar por requisições vindas de seu cliente. Quando uma delas recebe uma requisição para enviar uma mensagem na sala, ela encaminha essa mensagem diretamente para os clientes destinatários usando os respectivos sockets, sem envolver as outras threads. Assim, mais de uma thread pode querer escrever em um socket ao mesmo tempo, o que causaria corrompimento das mensagens. Logo, deve ser controlado o acesso a escrita em socket e por isso um mutex é utilizado.

A estrutura de dados que representa as salas contém referências aos clientes contidos nelas. Como mais de uma thread representando clientes distintos podem querer alterar essas estruturas e.g. requisições concorrentes de entrada em uma mesma sala, devem ser empregadas aqui também primitivas de sincronização. Assim, o controle a lista de clientes em uma sala, por exemplo, é serializado através de um Mutex pertencente àquela sala. Além disso as várias instâncias dessas salas são armazenadas em um array. Para controlar o acesso a esse array, o qual pode ser feito a partir de qualquer thread, um outro Mutex é utilizado.

3 Formato de Mensagens

Para definir a comunicaçao entre cliente e servidor, foi definido um formato de mensagem:

0	32
type	length
sequence number	data (>=0 bytes)
...	

Para cada request disponibilizado, foi criado um tipo de mensagem específico. Por exemplo, existe o tipo chamado de MESSAGE_NEW_NICK que carrega no campo dados o novo nick que o usuário deseja. Nota-se que o uso ou não desse campo de dados depende do tipo da mensagem.

4 Arquitetura do Cliente

O cliente é dividido em duas threads. Uma responsável pela interface com o usuário, o envio das requests ao servidor e o processamento das devidas respostas. A outra lê todas as mensagens enviadas pelo servidor. Ao ser recebida, a mensagem é adicionada a uma lista dinâmica da qual a outra thread pode ler as mensagens de resposta de seus requests. Essa mesma thread que recebe todas as mensagens é responsável por exibir na tela as mensagens vindas de outros clientes as quais foram encaminhadas pelo servidor.

Para o controle de acesso da lista de mensagens é usado um Mutex. Para evitar busy waiting por mensagens, uma variável de condição é usada para sinalizar a thread de UI que uma nova mensagem chegou para ela. Vale lembrar que as mensagens contendo a correspondência vinda de outros usuários, é processada diretamente assim que recebida pela mesma thread que recebe (que não é a de UI) e não é adicionada a lista de mensagens.

Usou-se um modelo concorrente devido ao fato de as mensagens poderem ser dividas em dois grupos: Mensagens de controle (as quais são requisições e respostas entre o servidor e o cliente que são síncronas) e mensagens de correspondência as quais são assíncronas, pois, o cliente não recebe confirmação explícita da aplicação. Apenas no nível de transporte.

Se o cliente envia uma requisição para mudar de nome de usuário, ele espera a resposta a essa requisição e não uma mensagem de outro cliente. Essa divisão entre threads permite que tenha mensagens síncronas e assíncronas usando a mesma conexão. Nota-se que todas as mensagens são síncronas a nível de sockets mas permite-se um comportamento assíncrono.

5 Dificuldades de Implementação

A principal dificuldade foi a implementação do protocolo de comunicação. Principalmente, garantir que os campos das mensagens fossem enviados corretamente.