Universidade Federal do Rio Grande do Sul
Instituto de Informática
INF01151 - Sistemas Operacionais II N (2015/1)
Turma B - Professor Dr. Alberto Egon Schaeffer Filho

Marcelo Souza Vasques - 208895

Relatório do Trabalho 3

- Ambiente de Testes:

O trabalho foi inteiramente desenvolvido em uma máquina rodando o sistema operacional Deepin, baseado no Ubuntu 14.04, em uma instalação 64 bits. Os testes foram executados nesse mesmo computador, que tem como especificações um processador Intel Core™ i5-4200U CPU @ 1.60GHz x 4, com 8 GB de memória RAM. O compilador utilizado para gerar o executável foi o GCC versão 4.8, com uso das bibliotecas *pthreads* e *GTK+ 2*.

- Concorrência:

Uma das características básicas do enunciado que define o trabalho é garantir a concorrência de tarefas no servidor. Para esta implementação, foram utilizadas threads que tratam cada conexão a um novo cliente separadamente. Cada thread mantém o contato com o socket até que o cliente feche a janela de sua aplicação, encerrando assim a thread responsável pela conexão dentro do servidor.

O número de threads definidas no servidor é fixo, fator que foi influenciado pelo tamanho da fila de espera de um socket existente em sistemas UNIX. Essa arquitetura utiliza, geralmente, filas de espera para quando existem mais do que cinco conexões TCP executando, estabelecidas pelo comando *listen*. Logo, utilizar cinco threads para realizar as conexões com os servidores ativos parece apropriado.

Afim de garantir maior autonomia e segurança dos dados para cada thread, elas possuem cinco buffers separados, nos quais são colocados os dados a serem transmitidos para os sockets. Nenhum thread acessa o buffer de outra, garantindo assim a segurança das mensagens.

Já no cliente, a situação é um pouco mais simples. A execução ocorre com a concorrência de duas threads, que compartilham o mesmo buffer, usado em situações de envio e recebimento de mensagens. Apesar de o envio ser limitado ao tempo que o usuário leva para digitar uma nova mensagem, o recebimento é feito por meio de intervalos de tempo pré-definidos (*timeouts*), que determinam a atualização do buffer caso haja conteúdo a ser recebido.

Logo, essa especificação claramente necessita de uma variável de controle de exclusão mútua para garantir a segurança do buffer. Isso é feito por meio de um *mutex* definido dentro da biblioteca de interface gráfica utilizada.

- Estruturas de dados:

As três estruturas principais existentes dentro do servidor são caracterizadas por listas. Para tal, cada umas das listas possui suas próprias definições de nodos, correspondentes às listas de usuários, salas e mensagens. Durante toda a existência do servidor, essas estruturas permanecem ativas, sendo alimentadas pelas mensagens recebidas dos clientes com diferentes dados.

As estruturas passadas para cada thread, por sua vez, guardam dois tipos de informações, o número do socket ao qual a thread está respondendo no momento e o buffer definido para que possam ocorrer os comando de *read* e *write*. Os buffers, finalmente, carregam quatro tipos de dados: a ID do usuário que enviou a mensagem, um inteiro que define se o usuário ainda está conectado, o nome do usuário em uma string e a mensagem enviada, limitada em 140 caracteres.

No cliente, a mesma estrutura de buffer é utilizada para que ocorra a troca de mensagens por meio dos sockets. Apenas dois dados são guardados localmente na aplicação de cada usuário, que são o número de socket obtido após a primeira tentativa de conexão e a ID de usuário definida pelo servidor, informada na primeira resposta vinda do socket após o envio da mensagem pelo servidor.

- Sincronização:

No servidor, afim de garantir a segurança das mensagens, como já mencionado, são utilizado buffers diferentes para cada thread. Dessa forma, já que cada *read* é bloqueante, o que acabou dificultando o processo de exclusão mútua sem, acidentalmente, estabelecer uma situação de *starvation*, essa abordagem surge como uma garantia para que os buffers não seja sobreescritos por threads distintas.

Contudo, a integridade das listas que guardam os usuários, as salas e as mensagens precisam estar funcionando corretamente para que nenhum cliente receba dados desatualizados ou corrompidos. Assim, três semáforos trabalham em conjunto, garantindo a segurança de cada uma das estruturas.

A sincronia de envio e recebimento de mensagens para usuários conectados a uma mesma sala ocorre no momento em que um usuário envia uma nova mensagem para o servidor. Caso o conteúdo não corresponda a um comando definido para a realização de funcões adicionais, os dados recebidos são imediatamente transmitidos de volta ao usuário e a todos os outros conectados à mesma sala, para impressão na caixa de mensagens.

O processo ocorre por ordem de chegada, ou seja, se uma mensagem chegar junto a outra, aquela que chegou primeiro irá ser enviada primeiro, separadamente em cada socket.

No cliente, por outro lado, o funcionamento ocorre com duas threads distintas que compartilham o mesmo buffer. Uma das threads escreve no buffer o conteúdo das mensagens escritas pelo usuário, enquanto a outra recebe os dados vindos do socket. Para garantir a integridade das mensagens, é usada uma *lock* do sistema de exclusão mútua definido pela biblioteca GTK+ 2.

- Adicionais:

O trabalho conta com uma interface de usuário gráfica amigável, desenvolvida com a biblioteca GTK+2. Em uma das duas caixas de texto principais, o usuário escreve suas mensagens e as envia. Enquanto na outra, utilizada somente para visualização, as diferentes respostas do servidor com outras mensagens de usuários e explicações dos comandos executados são exibidas.

Além dos comando de */join*, que cria uma nova sala caso ela não exista ou que conecta o usuário a uma sala já existente, e de */username*, pelo qual o usuário pode modificar seu nome, há também o comando */help*, que informa ao cliente os comando disponiveis.

Outro detalhe importante ocorre quando um usuário entra em uma sala já existente, na qual diferentes mensagens já podem ter sido enviadas. Definindo um ponto de permanência de dados, esse novo usuários recebe todas as mensagens enviadas antes de sua chegada. Por último, ao fechar a janela do chat, o sistema automaticamente lida com a saída do usuário, encerrando sua conexão corretamente.

- Compilando:

Em um sistema UNIX, com a biblioteca GTK+ 2 utilizada para desenvolvimento, um arquivo executável de cada uma das aplicações de cliente e de servidor podem ser obtidos por meio dos comandos listados abaixo.

- Servidor:
    - gcc -pthread -g -o server server.c list.c
- Cliente:
    - gcc -Wall -g client.c interface.c -o client `pkg-config --cflags gtk+-2.0` `pkg-config --libs gtk+-2.0`