UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA APLICADA

INF01151 – SISTEMAS OPERACIONAIS II N

TRABALHO 3: SISTEMA DE CHAT

INTEGRANTES:
BRUNO GABRIEL FERNANDES
MARCELO DE OLIVEIRA ROSA PRATES
SUYÁ PEREIRA CASTILHOS

1. Introdução

Neste trabalho, detalharemos a implementação de um sistema de Chat composto de uma aplicação Servidor e uma aplicação Cliente.

Diversas instâncias da aplicação Cliente deverão se conectar ao Servidor, de modo a implementar uma sala de troca de mensagens, em que cada mensagem enviada por um Cliente deverá ser re-enviada para todos os outros Clientes conectados.

O Servidor deverá ser implementado de modo a manter diversas conexões com módulos Cliente, e deverá tratar essas conexões de modo concorrente.

2. Ambiente de teste

Como ambiente de Teste, foi utilizado uma máquina virtual com auxílio do Oracle VM VirtualBox, com 4 cores e 512 mb de memória, executando o Sistema Operacional Ubuntu 11.04, tudo isso em cima de um Hospedeiro Intel I3 de 2 cores físicos e 2 cores virtuais com 4 gb de memória.

Como compilador, foi utilizado o GCC na versão 4.5.2.

3. Módulo Servidor

3.1. Estruturas

Foi utilizada uma estrutura que contém os dados de cada cliente conectado, que contém os seguintes campos:

- int online: um indicador que sinaliza se o cliente está conectado ou não.
- int id: o código de identificação do cliente.
- int socket: descritor do socket de comunicação com este cliente.
- char * username: o nickname do usuário conforme aparece nas suas mensagens de chat.

- pthread_t thread: a *thread* de execução que controla a comunicação com este cliente.

3.2. Inicialização

Na inicialização é feita a alocação prévia de espaço em memória para a lista de clientes conectados simultaneamente com o servidor.

Nesse passo são inicializadas duas threads que executam o comportamento do servidor, **readAdminCommands** e **acceptLoop**.

3.3. readAdminCommands

Esta *thread* executa lendo linhas de comando do terminal que executa o Servidor, ela espera por um comando *close* que sinaliza os demais fluxos de execução que o servidor deverá ser encerrado.

Quando essa sinalização é disparada, todas as conexões com os clientes são terminadas, e o servidor encerra a execução. Além disso, todos os clientes são notificados com a mensagem "KILL" (a ser interpretada pelo cliente), alertando-os para que finalizem sua execução e alertem o usuário do fechamento do chat.

3.4. acceptLoop

Esta *thread* fica em loop esperando conexões de módulos clientes, e quando recebe uma conexão ele inicializa uma thread que é responsável por tratar as conexões dos clientes, chamada **worker**.

3.5. worker

Esta *thread* inicialmente espera por que o cliente envie uma mensagem informando seu username, seu nome que irá aparecer como prefixo em todas as mensagens para os demais clientes.

Em seguida, ela entra em um loop em que fica esperando mensagens do cliente. Ao receber uma mensagem, ele verifica se ela é a de "logout". Se for a mensagem de "logout", ele encerra a conexão com o cliente, sinaliza que o cliente não está mais online (enviando uma mensagem no modelo "<username> LOGGED OUT" para cada cliente) e fecha a thread de execução.

Todas as demais mensagem recebidas são anexadas ao *username* do cliente ("<username> says: <message>") e enviadas a todos os clientes que estão online.

4. Módulo Cliente

O módulo cliente é responsável por conectar-se ao servidor, por meio de uma porta pré-determinada. Em seguida, ele se responsabiliza por capturar o username do usuário e repassá-lo para o servidor. Após isso, o módulo se encarrega de, concorrentemente, ler as mensagens do usuário (via standard input) e atualizar a interface de usuário de acordo com as

novas mensagens que vem sendo recebidas do servidor.

A fim de tratar as duas tarefas (leitura e escrita) de maneira concorrente, duas threads são disparadas: “reader” e “writer”.

4.1. reader

Esta thread executa um procedimento que consiste num loop em que a primitiva “read” é chamada iterativamente. Cada mensagem lida do servidor é concatenada ao fim de uma string que salva o log do chat do usuário. Ao fim, é executado um “refresh” na tela, em que o log antigo é substituído pelo log atualizado, tomando-se o cuidado de concatenar o “typing buffer” (o buffer que salva temporariamente a stdin do usuário) ao fim dele.

4.2. writer

Esta thread executa um procedimento que consiste num loop em que o stdin do usuário é capturado, salvo num buffer temporário e enviado para o servidor por meio da primitiva “write”. Antes disso, porém, é executado um “refresh” na interface do usuário para que o log do chat seja atualizado com novas mensagens.

4.3. logout e kill

Além das tarefas básicas, o módulo cliente se responsabiliza por parte do tratamento dos comandos de usuário e servidor. Ao verificar que o usuário digitou “logout”, o programa executa o fechamento do socket de comunicação com o servidor e termina a sua execução, tomando o cuidado de exibir a mensagem “you are now offline” (“você agora está offline”) na interface do usuário. Da mesma maneira, ao verificar que a mensagem recebida do servidor é “KILL”, o programa se responsabiliza por executar as mesmas operações do caso anterior.

5. Problemas encontrados

Durante o período de desenvolvimento, algumas dificuldades foram enfrentadas, no formato de defeitos no programa. Segue abaixo uma listagem destes problemas e das soluções empregadas para eliminá-los do sistema.

Problema:

Ao atualizar o log do chat de um usuário, o stdin (o que o usuário estava digitando) é perdido (desaparece da tela). Isso acontecia pois a rotina de “refresh” da interface do usuário é implementada com um system(“clear”) seguido da impressão (“print”) de um buffer que salva o log atualizado das mensagens do cliente. O system(“clear”) apaga toda a tela, inclusive o que o usuário poderia estar digitando. Assim, sempre que uma mensagem de um usuário era enviada durante a redação da mensagem de outro usuário, o segundo usuário deixava de enxergar o que tinha redigido até aquele ponto.

Solução:

Salvar o “stdin” do usuário num buffer e concatenar esse buffer ao fim do log do chat a cada atualização (“refresh”).

Problema:

O fechamento de uma aplicação cliente através de CTRL+C gera comportamentos anômalos no sistema, pois o servidor não tem conhecimento do fato de que o socket daquele cliente não deve ser lido. A tentativa do servidor de prover serviço a um cliente que não existe fazia com que o sistema entrasse num estado de erro e fosse fechado preemptivamente.

Solução:

Disponibilizar o comando de usuário “logout” para os clientes, permitindo que a aplicação seja finalizada de maneira controlada e segura. Ao logout de um cliente, todos os seus colegas recebem a mensagem “<username> LOGGED OUT”.

Problema:

O fechamento da aplicação servidor gera comportamentos anômalos nas aplicações cliente, pelos mesmos motivos citados no problema anterior.

Solução:

Disponibilizar o comando de usuário “close” para o servidor. Ao capturar esse comando, o servidor se encarrega de fechar as conexões com todos os clientes e comandar que cada um deles termine a sua própria execução.

5. Funcionalidades extras

Além das funcionalidades básicas, algumas adições foram feitas ao programa. Segue

abaixo uma listagem das funcionalidades adicionadas.

- Comandos de usuário
    - close:
        A aplicação servidor pode disparar o comando “close” para fechar todas as aplicações cliente, bem como terminar a sua própria execução.
    - logout
        A aplicação cliente pode disparar o comando “logout” para sair do chat. Os demais clientes serão alertados disso com uma mensagem no modelo “<username> is offline”.
    - buzz
        A aplicação cliente pode disparar o comando “buzz” para chamar atenção dos demais participantes do chat. Eles serão alertados com uma mensagem no modelo “<username> sent a buzz” e será executado um som característico.
- Sons
  O programa tem a possibilidade de tocar sons em alguns momentos. Para tal, o servidor deve ser executado com “./server sounds”. A execução de “./server” abre o servidor sem sons. Entre os sons tocados pelo programa, consta o som de recebimento de mensagem, o som de entrada no chat e o som de “buzz” (chamar atenção).

- Cores
  O programa associa uma cor a cada usuário, para facilitar a legibilidade do log de mensagens. Assim, toda vez que o username de um usuário for exibido, ele será impresso na tela com uma cor específica.

- Horário
  Todas as mensagens do programa são acrescidas do seu horário de recebimento, como em “23:1:53 fulano says oi”.

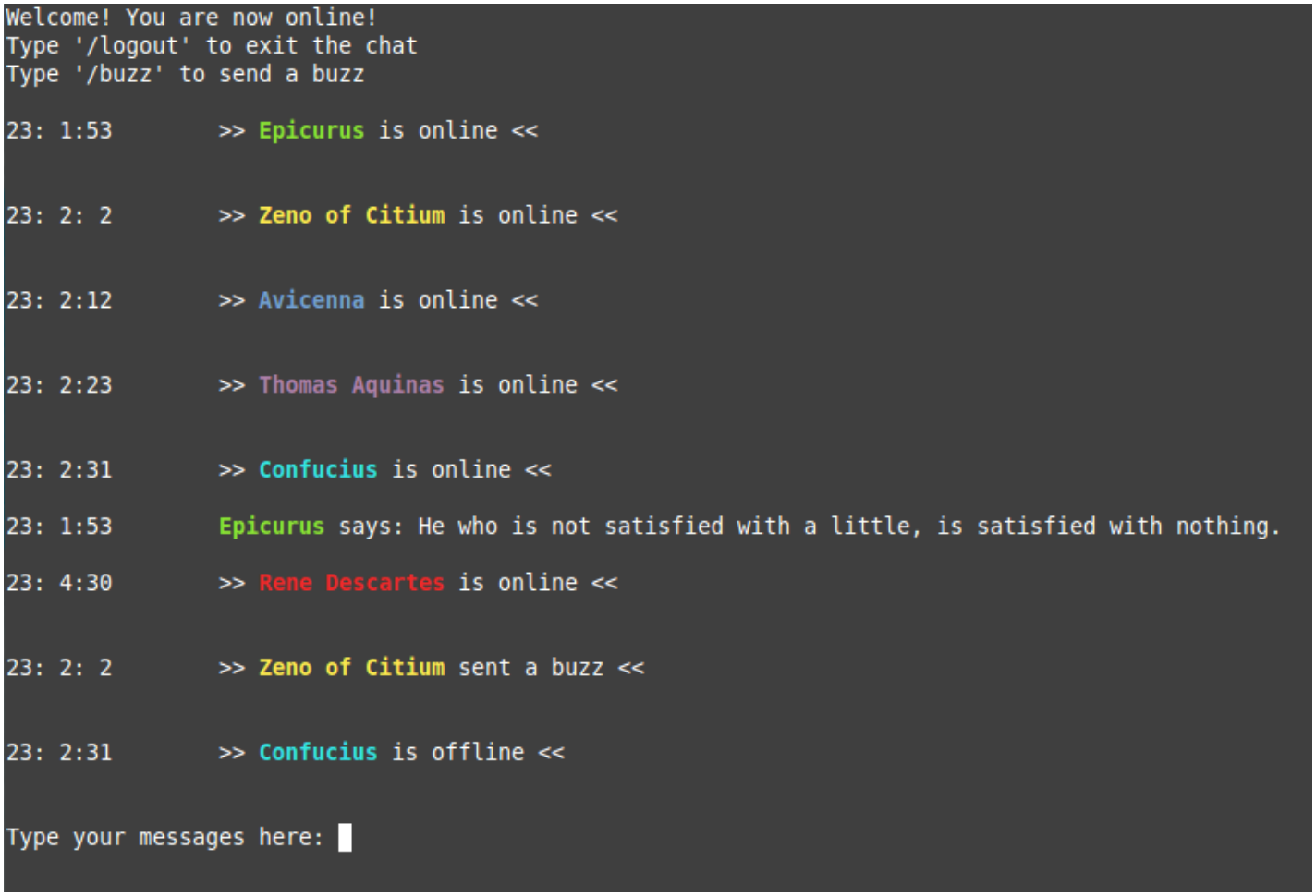

Imagem 1: screenshot da aplicação