Universidade Federal do Rio Grande do Norte

Departamento de Engenharia de Computação e Automação

DCA0130 – Redes de Computadores

Trabalho prático 2 – Criação de uma aplicação de bate-papo usando sockets

Entrega: 12/11/2019 23h59

Instruções:

- Trabalho em duplas;
- A nota deste trabalho será parte da avaliação da Unidade II;
- O trabalho consistirá na implementação de uma aplicação de bate-papo usando sockets;
- A submissão do trabalho deverá ocorrer via SIGAA até a data indicada acima;
- Não será necessária a elaboração de relatório, porém o trabalho deverá ser apresentado pessoalmente ao professor em horário a ser agendado;

Objetivo:

Colocar em prática os conhecimentos adquiridos em sala de aula e praticados em laboratório sobre as camadas de aplicação e de transporte e seus protocolos relacionados. Vocês irão criar uma sala de bate-papo utilizando sockets no modelo cliente/servidor. A implementação deverá ser em linguagem Python. Estão disponíveis scripts de exemplo como base para a implementação de sockets e threads em:

http://www.dca.ufrn.br/~viegas/disciplinas/DCA0130/files/Sockets/

Forma de avaliação e recomendações:

- Cada dupla irá apresentar e explicar o código fonte/implementação e o funcionamento da aplicação;
- A aplicação desenvolvida será executada em um computador (servidor) e outros computadores/dispositivos (clientes) irão se conectar ao mesmo;
- Cópias não são admitidas! Evitem usar implementações de outras pessoas, sob a pena de zerar a nota;
- Não é necessária a criação de uma interface gráfica. Não haverá qualquer bonificação na nota caso seja criada;
- O trabalho deverá estar com todas as funcionalidades implementadas e funcionando corretamente, sob pena da nota ser reduzida.

Instruções de funcionamento:

Criar uma aplicação cliente e outra servidora, em que:

1. O servidor é "uma sala" de bate-papo em grupo, onde os clientes irão se conectar (não havendo qualquer limite de clientes conectados).
2. Como protocolo de transporte deve ser utilizado o TCP.
3. Os clientes conectados poderão enviar e receber mensagens para o servidor (ou seja, modelo de comunicação cliente/servidor).
4. Quando um cliente envia mensagens para "a sala", o servidor deverá encaminhá-las para todos os demais clientes conectados (*broadcast*), bem como mostrá-las em sua própria tela (via

terminal). Neste ponto, tanto os clientes quanto o servidor devem exibir as mensagens nas suas respectivas telas.

5. Quando um cliente se conecta ao servidor, este solicitará um apelido (*nickname*), que será utilizado para identificar o cliente na comunicação:
    a. Quando um cliente se conecta ao servidor, deverá ser apresentada uma mensagem (para todos) de que o mesmo entrou na sala: `nick-do-cliente entrou...`
    b. Da mesma forma, quando um cliente sair da sala: `nick-do-cliente saiu!`
    c. Quando um cliente enviar uma mensagem, deverá ser apresentado em tela da seguinte forma: `nick-do-cliente escreveu: mensagem`
6. O servidor deverá exibir uma lista com os clientes a ele conectados, mostrando <NOME, IP, PORTA>. Esta informação pode ser solicitada tanto na tela do cliente (e exibida na mesma), quanto na tela do servidor (também exibida na mesma).
7. Um cliente, além de se conectar ao servidor para um bate-papo em grupo, também deve ser capaz de enviar mensagens privadas a outro cliente:
    a. Esta comunicação privada deve ter como intermediário o servidor, ou seja, o tráfego do cliente 'A' é enviado para o servidor e este encaminha para o cliente 'B' (e vice-versa);
    b. Se um cliente estiver conectado ao servidor de bate-papo em grupo e em seguida enviar uma mensagem privada para outro cliente, as mensagens trocadas entre esses clientes **não** devem ser exibidas na tela do servidor;
    c. Quando um cliente 'A' receber uma mensagem privada, deverá haver uma indicação de que tal mensagem é privada vinda do cliente 'B', por exemplo: (Cliente B enviou em privado: "oi").
8. Para que o servidor seja capaz de receber e responder às conexões de múltiplos clientes, o mesmo deverá ser implementado utilizando **threads**. Neste caso, para cada cliente que se conectar ao servidor deverá ser criada uma ou mais threads para gerenciar o envio e/ou o recebimento das mensagens. O objetivo das threads é permitir que várias funções no código possam ser executadas de forma paralela, sem causar bloqueios na execução. Como sugestão, quando um cliente se conectar ao servidor pode-se criar uma thread para recebimento de mensagens (`.recv()`) e outra para envio de mensagens (`.send()`). Lembrando que, os métodos "bloqueantes" na comunicação via sockets são: `.recv()`, `.input()` ou `.raw_input()`, `.accept()`.
9. A comunicação entre cliente e servidor deverá ser regida por um protocolo de aplicação, a ser desenvolvido. Cada mensagem trocada entre o servidor e o cliente deve conter um cabeçalho com os campos abaixo. As mensagens devem ser criadas neste formato, enviadas por meio do socket e interpretadas por quem as recebe.

2 octetos	16 octetos	8 octetos	variável
Tamanho da mensagem	*Nickname* do usuário	Comando	Dados

    a. Tamanho da mensagem: deverá conter o tamanho (em bytes) da mensagem a ser enviada, incluindo o cabeçalho;

b. *Nickname* do usuário: deverá indicar o *nickname* do usuário de destino. Para os casos em que o destino é "para todos" ou "para o servidor" fica a seu critério a escolha do que deverá conter este campo;
c. Comando: indica qual é o comando enviado (ex: `lista()`, `sair()`, `privado()`, etc.).
d. Dados: conteúdo da mensagem.

Instruções específicas (resumo):

1. O **servidor** deve:
   a. Aceitar conexões dos clientes (sem limites);
   b. Criar uma ou mais threads para cada cliente conectado;
   c. Solicitar um nome (*nickname*) a cada cliente que se conectar;
   d. Permitir que todos os clientes conectados enviem e recebam mensagens;
   e. Permitir que os clientes possam iniciar comunicações privadas por meio do comando `privado(*)`, onde `*` será o nome do cliente;
   f. Exibir a lista de clientes conectados por meio do comando `lista()`, mostrando o <NOME, IP, PORTA> de cada um;
   g. Encerrar todos os clientes quando o servidor for encerrado utilizando o comando `sair()`.
2. O **cliente** deve:
   a. Conectar-se ao servidor;
   b. Escolher um nome (*nickname*) para se identificar no bate-papo;
   c. Enviar e receber mensagens em grupo;
   d. Enviar mensagens privadas por meio do comando `privado(*)`, onde `*` será o nome do cliente a receber;
   e. Requisitar a lista de clientes por meio do comando `lista()`;
   f. Encerrar a sua aplicação ao digitar `sair()`.