Trabalho 2 de Redes de Computadores 2

Prof. Alessandro Vivas Andrade

Entrega e Avaliação:

1. **Formato:**
    a. Código-fonte organizado em diretórios separados para cada exercício.
    b. O upload do código deverá ser feito no Github.
2. **Documentação:**
    a. Comentários no código são obrigatórios em todos os scripts.
    b. Cada script ou log deverá conter o nome dos participantes.
3. **Critérios de Avaliação:**
    a. Funcionalidade e correção do programa.
    b. Uso adequado de threads e manipulação de sockets.
    c. Tratamento de erros e validação de entradas.
    d. Clareza e organização do código.
4. **Regras Gerais:**
    a. Trabalhos com erros de compilação ou execução serão anulados.
    b. Trabalhos copiados da Internet ou de colegas serão anulados.
    c. Plágios ou cópias serão anulados.
    d. Grupos de até 3 discentes.
    e. Todos os alunos deverão fazer upload do trabalho.
    f. Penalização de 3 pontos por dia de atraso.
5. **Requisitos Técnicos:**
    a. Utilizar Python 3.10 ou superior.
    b. Desenvolver e testar em um Sistema Operacional Linux.

Objetivo Geral

Desenvolver habilidades práticas de programação em redes de computadores utilizando os protocolos TCP e UDP. Os exercícios cobrem os principais conceitos de comunicação cliente-servidor, manipulação de threads e criação de um sistema de chat.

Exercício 1: Cliente-Servidor (TCP)

Objetivo:

Criar um cliente e um servidor que se comuniquem utilizando o protocolo TCP, permitindo o envio e recebimento de mensagens.

Instruções:

1. **Servidor:**
    a. Deve estar ativo em uma porta específica (ex: 5000).
    b. Deve aceitar conexões de múltiplos clientes.
    c. Ao receber uma mensagem do cliente, deverá imprimir no console e responder com uma confirmação (ex: "Mensagem recebida").
2. **Cliente:**
    a. Deve conectar-se ao servidor na porta especificada.
    b. Enviar uma mensagem ao servidor.
    c. Exibir a resposta do servidor no console.
3. **Requisitos Adicionais:**
    a. Implementar um mecanismo de validação para mensagens vazias.
    b. Garantir o encerramento seguro da conexão ao finalizar a comunicação.

Exercício 2: Servidor Echo (UDP)

Objetivo:

Criar um cliente e um servidor que utilizam o protocolo UDP para comunicação, implementando um serviço de eco, onde as mensagens enviadas pelo cliente são retornadas pelo servidor.

Instruções:

1. **Servidor:**
    a. Deve ouvir em uma porta específica (ex: 6000).
    b. Ao receber uma mensagem do cliente, deve enviá-la de volta (eco).
    c. Imprimir cada mensagem recebida no console.
2. **Cliente:**
    a. Deve enviar mensagens ao servidor e exibir a resposta recebida (eco).
    b. Permitir o envio contínuo de mensagens até a saída manual do usuário (ex: digitar "sair" para encerrar).
3. **Requisitos Adicionais:**
    a. Validar o tamanho máximo de mensagem permitido pelo UDP (64 KB).
    b. Tratar erros de comunicação (ex: perda de pacotes, tempo limite).

Exercício 3: Chat em Rede (TCP)

Objetivo:

Implementar um sistema de chat em tempo real que permita a comunicação bidirecional entre dois usuários.

Instruções:

1. **Servidor:**
    a. Aceitar conexões de dois clientes.
    b. Transmitir mensagens de um cliente para o outro em tempo real.
2. **Cliente:**
    a. Permitir que o usuário envie mensagens e visualize as mensagens recebidas simultaneamente.
3. **Requisitos Adicionais:**
    a. Usar threads para permitir o envio e recebimento de mensagens em paralelo.
    b. Implementar um comando de saída (ex: digitar "sair" para encerrar o chat).

Exercício 4: Servidor de Hora com Threads (TCP)

Objetivo:

Criar um servidor multithread que forneça a hora atual para múltiplos clientes simultaneamente.

Instruções:

1. **Servidor:**
    a. Rodar em uma porta específica (ex: 7000).
    b. Utilizar threads para atender múltiplos clientes em paralelo.
    c. Enviar a hora atual no formato: "HH:MM:SS" ao receber uma solicitação.
2. **Cliente:**
    a. Conectar-se ao servidor e solicitar a hora.
    b. Exibir a hora recebida no console.
3. **Requisitos Adicionais:**
    a. Implementar logs no servidor para registrar cada solicitação recebida e atendida.
    b. Garantir que o servidor continue funcionando mesmo em caso de falha de um cliente.

Exercício 5: Utilizando o software Wireshark capture o tráfego HTTP gerado pelo seu navegador

1. Abra o Wireshark
2. Acesse um site em seu navegador (ex: www.google.com).
3. Pare a captura e filtre os pacotes por HTTP.
4. Analise os cabeçalhos HTTP para identificar o método de requisição (GET, POST, etc.), o URL acessado, os cabeçalhos de resposta e o conteúdo da página.

Exercício 6: Utilizando o software Wireshark capture o tráfego TCP gerado no estabelecimento de conexão

1. Abra o Wireshark
2. Acesse o servidor Web de computador local ou remoto utilizando o telnet na porta 80
3. Colete os dados de conexão e verifique os cabeçalhos do TCP observando o processo de estabelecimento e encerramento de conexão.

Exercício 7: Utilizando o software Wireshark capture o tráfego gerado através de uma consulta via DNS

1. Abra o Wireshark e inicie uma captura de pacotes.
2. Acesse um site em seu navegador (ex: www.google.com).
3. Pare a captura e filtre os pacotes por DNS.
4. Analise os pacotes DNS para identificar o processo de resolução de nomes, os servidores DNS utilizados e os registros DNS obtidos.

Exercício 8: Utilizando o software Wireshark capture o tráfego gerado pelo comando ping.

1. Abra o Wireshark e inicie a capture de pacotes
2. Utilize o comando ping teste algum computador em sua rede
3. Pare a captura e filtre os pacotes por ICMP

Exercício 9: Utilizando o software Wireshark capture o tráfego gerado pelo DHCP

4. Abra o Wireshark e inicie a capture de pacotes
5. Conecte e desconecte da sua rede
6. Pare a captura e filtre os pacotes por DHCP

Exercício 10: . Utilizando Python e Linux crie um Chat utilizando WebSockets.