ATIVIDADE 02 : IMPLEMENTANDO UM SOCKET UDP

Disciplina : Plataformas de Distribuição

Professor : Nelson Souto Rosa

Integrantes : Maria Gabriela Alves Zuppardo e Victor Brunno Dantas de Souza Rosas

Enunciado:

Implemente uma aplicação cliente-servidor usando socket UDP para coletar e exibir, em tempo quase real, métricas de desempenho de vários computadores. Cada cliente é instalado em uma máquina a ser monitorada e atua como um agente, reunindo periodicamente dados como (escolha apenas um deles para monitorar): uso de CPU (percentual de ocupação por núcleo e média geral), memória (total, utilizada, livre), ´disco (uso de espaço, taxa de leitura/escrita), e rede (taxa de upload/download, pacotes perdidos). Essas informações são enviadas em intervalos configuráveis (por exemplo, a cada 5 segundos) para o servidor, que mantém conexões persistentes com todos os agentes. Por fim, o servidor mantém uma lista de clientes conectados e armazena os dados em memória.

Implementação:

O projeto é composto por dois arquivos: client.py e server.py

server.py

O servidor UDP foi construído usando a biblioteca “socket” para gerenciar a comunicação de rede. Em vez de threads o servidor utiliza um loop principal para escutar e processar datagramas de dados de forma assíncrona , eliminando a sobrecarga de gerenciamento de conexões.

- **Armazenamento de Dados:** Para manter as métricas dos clientes, foi criado um dicionário chamado `clientes = {}`. A chave do dicionário é o endereço IP do cliente, e o valor é um dicionário contendo a última métrica e o horário de recebimento, o que permite o monitoramento de múltiplos clientes sem a necessidade de uma conexão persistente.

- **Inicialização do Servidor:** O servidor é iniciado a partir de um objeto socket criado com `socket.socket(socket.AF_INET, socket.SOCK_DGRAM)`, que especifica o uso do protocolo UDP. Ele é associado ao endereço `0.0.0.0` e à porta `5005` com o comando `sock.bind((HOST, PORT))`, tornando-o acessível a qualquer cliente na rede que se comunique por essa porta.

- **Processamento de Mensagens:** Dentro de um loop infinito (`while True`), o servidor aguarda a chegada de um datagrama com `sock.recvfrom(8192)`. Ao receber os dados, ele os decodifica de bytes para uma string e, em seguida, usa a biblioteca **`json`** para converter a string em um dicionário Python. Por fim, os dados são armazenados no dicionário ***`clientes`*** e exibidos no terminal.

  client.py

O cliente é responsável por coletar as métricas de desempenho e enviá-las ao servidor.

- **Coleta de Métricas:** As bibliotecas `socket`, `json`, `psutil` e `time` são importadas. O cliente utiliza `psutil` para coletar uma gama mais ampla de métricas do sistema, como **CPU**, **memória**, **disco** e **rede**.

- **Processamento dos Dados:** As métricas coletadas são organizadas em um dicionário e, em seguida, convertidas para uma string no formato JSON com `json.dumps()`. A string JSON é então codificada

em bytes (`.encode("utf-8")`) para ser transmitida pelo socket, um processo conhecido como *marshalling*.

- **Envio de Dados:** O cliente cria um socket UDP (`socket.socket(socket.AF_INET, socket.SOCK_DGRAM)`) e envia o datagrama de dados diretamente ao endereço e porta do servidor (`sock.sendto(mensagem, (SERVER_HOST, SERVER_PORT))`) sem a necessidade de estabelecer uma conexão prévia. Esse processo é repetido em um intervalo de 5 segundos, garantindo o monitoramento contínuo.

Comparando com o socket TCP :

Característica	**Socket (TCP)**	**Socket (UDP)**
Protocolo de Transporte	TCP (SOCK_STREAM) orientado à conexão, garantindo entrega confiável e ordenada	UDP (**`SOCK_DGRAM`**), sem conexão, focado em agilidade e eficiência, mas sem garantia de entrega ou ordem dos pacotes.
Arquitetura do Servidor	Servidor com threads dedicadas para cada cliente, aguardando conexões com `server.accept()`.	Servidor com um único loop, usando `sock.recvfrom()` para receber datagramas de qualquer cliente. Não há necessidade de gerenciar conexões persistentes.
Métricas Coletadas	Apenas métricas de CPU (uso por núcleo e média).	Conjunto expandido de métricas, incluindo **CPU, memória, disco e rede**.
Formato de Dados	Strings formatadas de forma simples.	json

Comunicação	Comunicação bidirecional e confiável. O cliente só envia dados após a conexão ser estabelecida	Comunicação "envia e esquece". O cliente envia dados para o servidor sem ter certeza de que o servidor está pronto para recebê-los, o que é aceitável para dados de monitoramento onde a perda de pacotes pode ser tolerada.