Projeto DNS Infraestrutura de Comnumicação

```
Este projeto teve como objetivo implmentar uma comunicação cliente-servidor via
UDP garantindo confiabilidade. Um servidor-DNS simplificado é responsável por
fornecer o endereço do servidor ao cliente, uma vez que ele já tenha previamente
recebido o endereço do servidor.

Autor: José Roberto Fonseca e Silva Júnior, jrfsj@cin.ufpe.br
```

Os 3 módulos criados para esse projeto foram:

- dns.py
- server.py
- client.py

Seguindo ordem cronológica, a execução e operação desse projeto se dá nos seguintes passos:

1. `dns.py` é executado. Ele fica num loop esperando por mensagens ou do servidor ou do cliente.
2. `server.py` é executado e manda seu domínio e endereço para o servidor DNS.
3. `client.py` é executado. Este fará uma requisição pelo endereço do domínio do server.
4. O cliente com o endereço em mãos, conecta-se com servidor via TCP.
5. Após ter realizado conexão TCP, esta conexão é fechada para, a partir de agora, só se comunicar via UDP com o servidor. Um leque de operações cliente-servidor é apresentado via console ao cliente. As operações são realizadas até que a comunicação seja terminada.

Segundo essa ordem de execução deles é preferível que seja primeiro o `dns`, segundo o `server`, terceiro o `client`.

As seções deste relatório foram divididas em 4:

- Módulo DNS: descreve a implementação do módulo DNS, seja sua comunicação com o cliente ou servidor, assim como o mapeamento dos endereços.
- Módulo servidor: descreve os passos realizados pelo servidos até se comunicar com o cliente.
- Módulo cliente: descreve as operações realizadas pelo cliente e a interface voltada usuário para comunicação cliente-servidor.
- Confiabilidade de UDP: apresenta a abordagem tomada para garantir confiabilidade do transporte de mensagens via UDP.

Módulo DNS

Mapeamento de endereços

O trabalho do DNS-server é mapear um nome de domínio para um endereço. Este processo pode ser expresso na estrutura de dados `dict`, do dicionário do Python, pois toma como entrada uma chave e retorar um valor associado a ela. No caso do módulo DNS, a chave é o nome do **domínio** e o valor é o **endereço**.

Seja o dicionário de endereços `dns_list`, uma forma de acessar o endereço do site *www.foo123.com* seria dessa forma:

dns_list["www.foo123.com"] -> ("localhost", 65432)

Inicialização

Ao ser inicializado, o módulo DNS cria um socket UDP, faz um bind no endereço **(localhost, 65431)**, oculto no macro **THIS_ADDR** e conhecido pelo cliente e sevidor.

```
with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:  # creating UDP socket
    print("DNS socket created")

    s.bind(THIS_ADDR)  # bind to ("localhost", 65431)
```

Loop

Usando o socket criado, o DNS-server entra num **loop eterno** enquanto aberto para receber mensagens.

Formato geral de uma mensagem

Uma mensagem recebida vem na forma `source;{};{};{}`, onde cada componente vem separado por ponto e vírgula (`;`).

- `source`: Indica quem é a fonte da mensagem. Pode ser ou `client` ou `server`, indicando que a mensagem vem do cliente ou do servidor, respectivamente.
- `{}`: Campos opcionais. Varia com o propósito de cada mensagem.

```
while True:
    data, addr = s.recvfrom(BUF)  # wait for data to receive
    msg = data.decode("utf-8").split(';')  # message handling
```

O DNS é preparado para receber dois tipos de mensagens, sendo uma delas vinda do servidor e outra vindo do cliente.

Mensagem do servidor

O servidor só se comunica com o DNS-server para mandar seu domínio e endereço, então se a mensagem recebida vier de um servidor, os campos opcionais `{}` virão na forma `{domain_name};{host};{port}`. Assim o DNS-server atualiza dicionário de relações com uma nova entrada.

```
if msg[0] == "server":
    key, value = (msg[1], (msg[2], msg[3]))
    dns_list[key] = value  # update dns dictionary
```

Mensagem do cliente

O cliente só se comunica com o DNS-server para requisitar um endereço baseado num nome de domínio. Então se a mensagem vier de um servidor, o módulo DNS checa se existe uma entrada no seu `dns_list` com esse nome. Se sim, ele retorna o valor associado (endereço), senão, retorna `"null"`.

```
elif msg[0] == "client":
    key = msg[1]

    if key in dns_list:
        reply = str(dns_list[key]).encode()
        s.sendto(reply, addr)  # reply client with server addres
    else:
        s.sendto("null".encode(), addr)  # domain name not in dict. Reply null to
client
```

Módulo servidor

Esta seção descreve as operações realizadas pelo servidor, que podem ser divididas em duas: **comunicação com servidor DNS** e **comunicação com cliente**.

Comunicação com servidor DNS

A primeira operação que o servidor realiza é mandar seu domínio e endereço para o servidor DNS via UDP.

```
def main():
    send_address_to_dns(DOMAIN, DNS_ADDR)

[...]

def send_address_to_dns(server_dns, dns_addr):
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
        # building message
        # "www.foo123.com;localhost;65432"
        msg = "server;{};{};{}".format(server_dns, THIS_ADDR[0],
THIS_ADDR[1]).encode()

        s.sendto(msg, DNS_ADDR)  # sending message to DNS-server
```

Comunicação com cliente

Após ter mandando seu endereço para o módulo DNS, é esperado que o cliente se comunique com o servidor. Para isso, o servidor cria um socket UDP associado com o mesmo endereço que foi mandado para o DNS-server.

```
def udp_with_client():
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
```

```
        s.bind(THIS_ADDR)

        while True:
            data, addr = s.recvfrom(BUF)
            msg_received = data.decode("utf-8")

            if msg_received == "1":  # client requested list of files
                msg = str(os.listdir("./server_data"))  # list of files in
./server_data
                s.sendto(msg.encode(), addr)  # send list to client

            elif msg_received == "2":  # client requested a file
                data, addr = s.recvfrom(BUF)
                send_file(data.decode("utf-8"), addr, s)

            elif msg_received == "0":  # client stopped communication
                break
```

Dentro do loop, o server age de acordo com a operação que o cliente deseja fazer. No total são 3 operações, e suas descrições estão seção *Módulo cliente* e *Garantindo confiabilidade*

Módulo cliente

Inicialização e comunicação com DNS-server

Para se comunicar com o servidor, o cliente primeiramente faz uma requisição ao servidor DNS pelo endereço de um certo domínio. Assim, primeiro comando executado pelo cliente é a chamada da função `ask_address_to_dns` que trata essa comunicação com o DNS-server.

```
def ask_address_to_dns(domain):
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
        s.bind(THIS_ADDR)

        msg = "client;{}".format(domain).encode()  # message building
                                                   # "client;www.foo123.com"

        s.sendto(msg, DNS_ADDR)  # send dns request message

        data, addr = s.recvfrom(BUF)  # await for response

        return handle_dns_message(data)
```

A função exposta acima retorna o endereço do servidor. *OBS Está oculta a função `handle_message` para simplificar a explicação, mas o que ela faz é basicamente tratamento de string.*

Comunicação com servidor

Num primeiro momento, o cliente-servidor estabelecem uma conexão TCP. Depois, trocam mensagens apenas via UDP.

Um socket UDP é criado e associado ao endereço do servidor.

```
with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
    s.bind(THIS_ADDR)
```

Depois, este socket é usado num loop infinito que perdura até que o cliente deseje encerrar a comunicação.

```
while True:
    data, addr = s.recvfrom(BUF)
    msg_received = data.decode("utf-8")
```

Dentro do loop, a função `get_user_input` dispõe para o usuário 3 opções de ação:

0. End communication. Esta ação interrompe o andamento tando do loop após mandar uma mensagem para o servidor com o conteúdo "*0*", indicando para ele também interromper seu loop.
1. List files. Envia uma mensagem "*1*" ao servidor requisitando uma lista dos arquivos no banco de dados.
2. Request file. Primeiro, envia uma mensagem ao servidor com o número da operação ("*2*"), que, por sua vez, espera por outra mensagem. Essa segunda mensagem que vem do cliente contém o *filename*. Então, o cliente executa a função `receive_file`, que está melhor descrita na seção *Garantindo confiabilidade*

Garantindo confiabilidade