UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
COLEGIADO DE CIÊNCIA DA COMPUTAÇÃO

Relatório Implementação de Cliente e Servidor NTP

Professor: Luiz Antonio Rodrigues
Alunos: Gustavo Orlandini, Paulo Scalon

- Protocolo NTP (Network Time Protocol)

O NTP é um protocolo de comunicação usado para sincronizar os relógios de computadores e dispositivos em uma rede, de forma precisa e eficiente. Ele permite que um cliente obtenha a hora de um servidor NTP, ajustando seu relógio para refletir o tempo preciso. O protocolo NTP opera em um modelo cliente-servidor e usa o conceito de timestamps para transmitir a hora.

O Cliente envia uma solicitação para o servidor NTP, incluindo um timestamp de quando a solicitação foi feita;

O servidor responde com seu próprio timestamp (quando ele recebe a solicitação e quando envia a resposta), assim como o timestamp enviado pelo cliente;

O cliente usa os timestamps recebidos para calcular:

Offset: A diferença entre o tempo real no servidor e o tempo do cliente.

Delay: O tempo de viagem dos pacotes entre o cliente e o servidor.

- Implementação Cliente NTP

O código do cliente NTP foi implementado usando a biblioteca `socket` para enviar e receber pacotes NTP via UDP. Ele também utiliza `struct` para empacotar e desempacotar os dados de acordo com o formato do NTP. O cliente possui uma interface gráfica simples feita com `Tkinter`, onde o usuário pode inserir o endereço de um servidor NTP enviando um pacote no formato cliente (modo 3) para o servidor e recebe uma resposta que contém a hora do servidor.

Principais funções:

`get_time(self):`

- Recebe o nome do servidor NTP inserido pelo usuário.
- Chama a função `request_time` para obter a hora, o offset e o delay.
- Exibe os resultados na interface gráfica.

```python
def get_time(self):
    server = self.entry.get()
    try:
        time_data, offset, delay = self.request_time(server)
        self.result_label.config(text=f"Horário: {time_data}")
        self.offset_label.config(text=f"Offset: {offset:.6f} segundos")
        self.delay_label.config(text=f"Delay: {delay:.6f} segundos")
    except Exception as e:
        messagebox.showerror("Erro", f"Não foi possível obter o tempo: {e}")
```

`request_time(self, server):`

- Estabelece uma conexão UDP com o servidor.
- Cria o **pacote de solicitação NTP** com o formato adequado (Modo 3: Cliente) utilizando `struct.pack`.
- Envia a solicitação e aguarda uma resposta do servidor.
- **Desempacota a resposta NTP** recebida do servidor.
- Calcula o **offset** (diferença entre o tempo do servidor e o tempo do cliente) e o **delay** (tempo de ida e volta do pacote).
- Exibe a hora ajustada para o fuso horário configurado.

Formato do pacote NTP (Modo 3 - Cliente):

O pacote enviado pelo cliente possui 48 bytes, onde:
1 byte para o indicador de leap (salto), versão e modo.
1 byte para o stratum (nível de precisão do servidor), intervalo de polling, etc.
Timestamps de referência, origem, recebimento e transmissão.

```python
def request_time(self, server):
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
        sock.settimeout(5)  # Timeout de 5 segundos
        addr = (server, NTP_PORT)

        # Construção do pacote NTP (Modo 3 - Cliente)
        solicitacao = struct.pack(
            "!B B B b 3I 8I",
            0b00100011,  # Leap Indicator (0), Versão (4), Modo (3 - Cliente)
            0,  # Estrato (não usado pelo cliente)
            0,  # Intervalo de Poll
            -6,  # Precisão (-6 ≈ 15.6ms)
            0, 0, 0,  # Root Delay, Root Dispersion, Reference ID
            0, 0,  # Timestamp de Referência (segundos, fração)
            0, 0,  # Timestamp de Origem (segundos, fração)
            0, 0,  # Timestamp de Recebimento (segundos, fração)
            0, 0   # Timestamp de Transmissão (segundos, fração)
        )
```

```python
    # Enviar solicitação NTP
    timestamp_origem = time.time()  # Tempo de envio da solicitação
    sock.sendto(solicitacao, addr)

    # Receber resposta
    dados, _ = sock.recvfrom(48)  # Receber resposta de 48 bytes
    timestamp_destino = time.time()  # Tempo em que a resposta chegou

    # Desempacotar a resposta NTP
    desempacotado = struct.unpack("!12I", dados)
    segundos_transmissao = desempacotado[10]  # Timestamp de Transmissão
    fracao_transmissao = desempacotado[11]  # Timestamp de Transmissão (fração)

    # Converter para tempo Unix
    timestamp_transmissao = segundos_transmissao - DELTA_TIMESTAMP_NTP + (fracao_transmissao / (2**32))

    # Calcular offset e delay
    offset = ((timestamp_transmissao - timestamp_origem) + (timestamp_transmissao - timestamp_destino)) / 2
    delay = (timestamp_destino - timestamp_origem) - (timestamp_transmissao - timestamp_origem)

    # Ajustar a hora para o horário correto (considerando o fuso horário)
    timestamp_local = timestamp_transmissao + (FUSO_HORARIO * 3600)

    # Formatando a hora local
    time_data = time.strftime('%Y-%m-%d %H:%M:%S', time.gmtime(timestamp_local))

    return time_data, offset, delay
```

- Implementação Servidor NTP

O servidor NTP também é implementado utilizando a biblioteca `socket`, com a diferença de que ele espera solicitações de vários clientes, processa essas solicitações e envia as respostas com a hora correta. O servidor escuta na porta *123*, que é a porta padrão para o protocolo NTP. O servidor recebe pacotes de solicitação NTP dos clientes, calcula a hora correta (usando o tempo Unix) e responde com um pacote no formato do **modo 4 (Servidor)**, que contém timestamps para os eventos de recebimento e transmissão.

Principais Funções:

`servidorNtp()`:

Cria o socket UDP e escuta na porta 123.

Quando uma solicitação chega, ele extrai o **timestamp de origem** (o momento em que a solicitação foi enviada pelo cliente) do pacote NTP.

Calcula o **timestamp de transmissão**, que é o tempo em que o servidor responde.

Converte todos os timestamps para o formato NTP, ajustando para o **epoch NTP (1900)**.

Monta o **pacote de resposta NTP (Modo 4 - Servidor)** e envia de volta ao cliente.

Formato do pacote NTP (Modo 4 - Servidor):

O pacote de resposta do servidor possui 48 bytes:
1 byte para o leap, versão e modo.
1 byte para o stratum (nível do servidor), intervalo de polling, etc.
Timestamps para a referência (servidor), origem (cliente), recebimento (servidor) e transmissão (servidor).

```python
def servidorNtp():
    # Cria o socket UDP
    servidor = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    servidor.bind(("0.0.0.0", NTP_PORT))  # Ouvindo em todas as interfaces na porta 123

    print(f"Servidor NTP em execução na porta {NTP_PORT}...")

    while True:
        # Espera por dados de um cliente
        dados, endereco = servidor.recvfrom(48)  # O tamanho do pacote NTP é fixo em 48 bytes
        timestampRecebido = time.time() + NTP_TIMESTAMP_DELTA  # Tempo de recebimento (NTP)

        # Extrai o timestamp de origem do cliente (de onde veio a solicitação)
        timestampOrigem = struct.unpack("!Q", dados[24:32])[0]
        timestampOrigem = timestampOrigem - NTP_TIMESTAMP_DELTA  # Ajusta para o epoch Unix

        # Ajusta o timestamp de origem para garantir que ele esteja dentro do limite
        if timestampOrigem < 0:
            timestampOrigem = 0

        # Calcula o timestamp de transmissão (tempo atual do servidor)
        timestampTransmissao = time.time() + NTP_TIMESTAMP_DELTA

        # Converte os timestamps para inteiros e fração de segundos
        secs_transmissao = int(timestampTransmissao)
        frac_transmissao = int((timestampTransmissao - secs_transmissao) * (2**32))

        secs_origem = int(timestampOrigem)
        frac_origem = int((timestampOrigem - secs_origem) * (2**32))
        secs_recebido = int(timestampRecebido)
        frac_recebido = int((timestampRecebido - secs_recebido) * (2**32))

        try:
            # Monta resposta NTP (modo 4 - servidor)
            resposta = struct.pack(
                "!B B B b 11I",
                0b00100100,  # Leap Indicator (0), Versão (4), Modo (4 - Servidor)
                1,  # Stratum (1 - Primário)
                0,  # Poll Interval
                -6,  # Precision (-6 ≈ 15.6ms)
                0,  # Root Delay
                0,  # Root Dispersion
                0x4C4C4F43,  # Reference ID (pode ser um identificador qualquer, como "LOCL")
                secs_transmissao, frac_transmissao,  # Reference Timestamp
                secs_origem, frac_origem,  # Origem (cliente)
                secs_recebido, frac_recebido,  # Recebimento (servidor)
                secs_transmissao, frac_transmissao  # Transmissão (servidor)
            )
        except struct.error as e:
            print(f"Erro ao empacotar resposta: {e}")
            continue

        # Envia a resposta ao cliente
        servidor.sendto(resposta, endereco)
        print(f"Respondendo ao cliente {endereco} com a hora: {time.ctime(timestampTransmissao)}")
```

- Problemas Encontrados e Respectivas Soluções:

Erro de timestamp de origem fora do limite:

Durante o empacotamento da resposta NTP, o valor de segundos de origem estava fora do intervalo esperado.

Solução: Implementar um ajuste no timestamp de origem, garantindo que ele estivesse dentro do intervalo válido. Quando o valor fosse negativo, ajustar para zero.

Erro ao empacotar a resposta:

Foi identificado um erro ao tentar empacotar os dados da resposta NTP devido ao formato incorreto dos timestamps.

Solução: Refatorar a conversão de timestamps para garantir que a fração de segundos fosse calculada corretamente e que os valores estivessem dentro dos limites do protocolo.

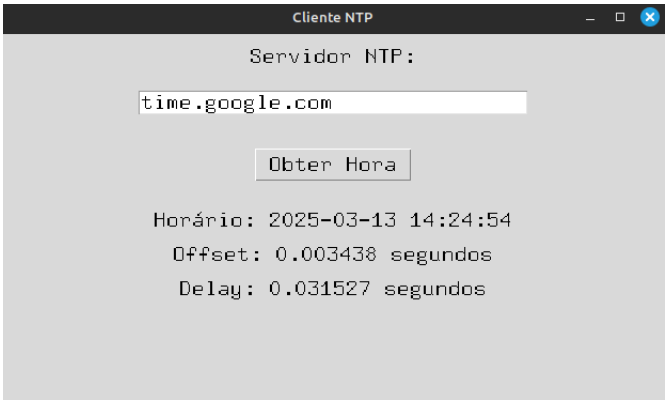
Erro de "argument out of range":

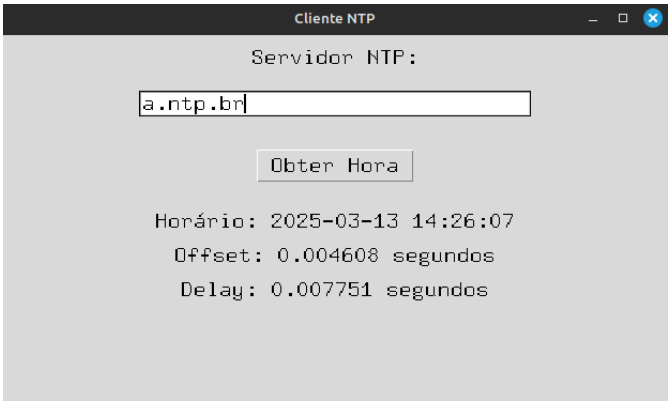
Quando o valor do timestamp estava fora do intervalo permitido, acusava o erro “argument out of range” ao tentar empacotar os dados.

Solução: Foram feitas verificações para garantir que os valores de segundos e fração de segundos estivessem dentro do intervalo de 0 a 232−1. Se o valor estivesse fora desse intervalo, o valor é corrigido ou descartado.

- Testes:

Testando Apenas o Cliente com servidores NTP oficiais:





Teste com o servidor local: