Relatório - Projeto Redes de Computadores

Aluno: Paulo Roberto Mendes dos Santos

Projeto de uma aplicação em redes para controle de telescópios usando sockets com protocolo IPv4. Este projeto é um estudo de funcionalidade e prova de conceito experimental para uma rede de telescópios ou observatórios onde vários cientistas podem realizar o controle do equipamento sem a necessidade de estar no local exato do telescópio.

Principais Funcionalidades

Controle de orientação em termos siderais: É possível ajustar a ascensão reta e a declinação do telescópio, para alinhar com um corpo celeste a que se deseja observar.

Controle de rastreamento: É possível iniciar o rastreamento sideral do objeto ao qual o telescópio está orientado.

O que poderia ser implementado a mais

Há inúmeros comandos que podem ser implementados, de forma simples, ao projeto final. Além disso, também poderia ser implementado um cliente observador, focado em receber todos os comandos enviados ao telescópio e exibir a condição do mesmo.

Outra opção seria a de criar abas dentro do lado do cliente para melhor separar os comandos que podem ser enviados.

Principais dificuldades na implementação do projeto

Entender como manter uma conexão aberta por meio de threads foi a maior dificuldade, além disso, organizar a interface de forma a enviar os comandos por meio de uma thread específica também apresentou-se como um desafio, já que os comandos deveriam ser executados por meio de callbacks de botões da interface, e ela roda por si só em uma thread do tk.

Entender como fazer o recv() “esperar” por um comando do cliente também foi uma dificuldade antes de ter as threads como solução.

A abordagem de um gerenciador de comandos que observa o estado de pedido de envios ao invés de enviar diretamente os dados foi a solução que se provou eficiente e de maior manutenção,

Durante o desenvolvimento, também foi notada a necessidade de testar as mudanças de forma frequente, pois pequenas alterações influenciavam a estabilidade do sistema de forma considerável.

Executando o projeto:

- Abra um terminal no local do projeto (./scopenet)
- Execute um servidor observatório.

```
$ cd scopenet
$ python scope-server.py
```

Noutro terminal, simule uma sessão de um cliente:

```
$ python scope-client.py
```

Uma interface deve aparecer agora, onde é possível inserir instruções para o observatório. Também é possível abrir mais de uma sessão cliente para controlar o observatório.

Há dois comandos disponíveis:

- `GoTo` envia o telescópio para a posição sideral desejada. (Lembre-se de preencher a posição em coordenadas equatoriais!)
- `Track` Inicia o acompanhamento do alvo atual.

Caso deseje criar uma seção entre computadores numa rede local, altere o endereço do `HOST` encontrado nos arquivos acima executados.

Código fonte: Servidor

```
"""
 :file: scope-server.py
 :author: Paulo Santos (pauloxrms@gmail.com)
 :brief: Servidor base, representando um observatório.
 :version: 0.1
 :date: 22-02-2024

 :copyright: Copyright Paulo R. Santos (c) 2024
"""

import socket
import json
import threading

HOST = '127.0.0.1'  # Endereço IP local
PORT = 65432  # Porta do servidor

target_ra = {"h": 0, "m": 0, "s": 0}
target_dec = {"°": 0, "'": 0, "\"": 0}

slewing = False


def save_target_loc(goto_metadata: dict[str, str]):
```

```
    """
    Atualiza a posição alvo do telescópio.

    :param goto_metadata: Dados obtidos pelo socket, represen
    :return: -1 em caso de erro, 0 caso contrário.
    """
    ra_recv = goto_metadata['ra'].split(" ")
    dec_recv = goto_metadata['dec'].split(" ")

    try:
        for i, k in enumerate(target_ra.keys()):
            target_ra[k] = int(ra_recv[i][:-1])

        for i, k in enumerate(target_dec.keys()):
            target_dec[k] = int(dec_recv[i][:-1])

    except ValueError:
        return -1

    slewing = True
    return 0


def process_command(command):
    """
    Trata o comando recebido.

    :param command: Comando recebido como dicionário json.
    :return: Resposta ao comando.
    """
    if slewing:
        return "Telescope is slewing. Wait to finish."
    if command["command"] == "TRACK":
        return ("Tracking current position:\n"
                f"RA: {target_ra};\nDEC: {target_dec}")
    elif command["command"] == "GOTO":
        if save_target_loc(command['metadata']) != 0:
            return f"Incorrect data:\nRA={command['metadata']
```

```
        else:
            return f"Going to:\nRA={command['metadata']['ra']
    elif command["command"] == "DISCONNECT":
        return "DISCONNECT"
    else:
        return f"Command '{command['command']}' not recognize


def handle_connection(client: socket):
    """
    Thread que gerencia uma conexão socket com um client.

    :param client: Cliente conectado.
    """
    connected = True
    while connected:
        try:
            # Recebe os dados do cliente
            data = client.recv(1024).decode()

            if data:
                # Decodifica os dados recebidos do cliente
                command_json = json.loads(data)
                print(f"[{addr[0]}:{addr[1]}]: {command_json[
                response = process_command(command_json)

                if response == "DISCONNECT":
                    connected = False
                    response = "Telescope detached"
                    connected_clients.remove(client)
                    print("Fim de conexão solicitado.")

                client.send(response.encode())
        except:
            connected_clients.remove(client)
            connected = False
            print("Conexão perdida.")
    client.close()
```

```
if __name__ == "__main__":
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((HOST, PORT))
    s.listen()

    connected_clients = []
    print("Telescope is waiting for connection...")

    while True:
        conn, addr = s.accept()
        if conn not in connected_clients:
            print(f"Connected to {addr[0]}! Waiting for comma
            connected_clients.append(conn)
            conn.send("Connected successfully.".encode())

            thread = threading.Thread(target=handle_connectio
            thread.start()
        else:
            conn.send("Already connected.".encode())
```

Código fonte: Cliente

```
"""
 :file: scope-client.py
 :author: Paulo Santos (pauloxrms@gmail.com)
 :brief: Cliente base, representando um controlador do observ
 :version: 0.1
 :date: 22-02-2024

 :copyright: Copyright Paulo R. Santos (c) 2024
"""

import socket
import json
import threading
```

```
import tkinter as tk
from tkinter import ttk

# Configurações do cliente.
HOST = '127.0.0.1'  # Endereço IP do servidor
PORT = 65432  # Porta do servidor

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))

command_requests = {
    "DISCONNECT": False,
    "GOTO": False,
    "TRACK": False,
}

commands = {
    "DISCONNECT": {
        "command": "DISCONNECT",
        "metadata": "NULL"
    },
    "GOTO": {
        "command": "GOTO",
        "metadata": {
            "ra": "",
            "dec": "",
        }
    },
    "TRACK": {
        "command": "TRACK",
        "metadata": "NULL"
    },
}


def on_track():
    command_requests["TRACK"] = True
```

```
def on_goto():
    commands["GOTO"]["metadata"]["ra"] = f"{ra_hour.get()}h {
    commands["GOTO"]["metadata"]["dec"] = f"{dec_degree.get()
    command_requests["GOTO"] = True


def on_dc():
    command_requests["DISCONNECT"] = True


def message_manager():
    """
    Thread responsável pelo gerenciamento de mensagens a sere
    """
    response = ("-" * 70) + "\n" + f"{s.recv(1024).decode()}"
    response_label.config(text=response)
    run = True
    while run:
        for cbr in list(command_requests.keys()):
            if command_requests[cbr]:
                try:
                    s.send(json.dumps(commands[cbr]).encode()
                    response = "Successfully sent command\n"
                except:
                    response = "Unable to connect with telesc
                    s.close()
                    run = False
                command_requests[cbr] = False
                response_label.config(text=response)


if __name__ == "__main__":
    root = tk.Tk()
    style = ttk.Style()
    style.configure('TButton', font=('Arial', 12), padding=5)
    root.title("Cliente ScopeNet")
    root.iconbitmap("assets/favicon.ico")
```

```
    separator = ttk.Separator(root, orient='horizontal')
    separator.grid(row=0, column=0, columnspan=2, sticky='ew'

    frame_ra = ttk.Frame(root)
    frame_ra.grid(row=1, column=0, padx=10, pady=5)
    ra_hour = tk.StringVar()
    ra_hour_entry = ttk.Entry(frame_ra, width=5, textvariable=
    ra_hour_entry.grid(row=0, column=0, padx=(0, 5))
    ttk.Label(frame_ra, text="h").grid(row=0, column=1, padx=
    ra_minute = tk.StringVar()
    ra_minute_entry = ttk.Entry(frame_ra, width=5, textvariab
    ra_minute_entry.grid(row=0, column=2, padx=(0, 5))
    ttk.Label(frame_ra, text="m").grid(row=0, column=3, padx=
    ra_second = tk.StringVar()
    ra_second_entry = ttk.Entry(frame_ra, width=5, textvariab
    ra_second_entry.grid(row=0, column=4, padx=(0, 5))
    ttk.Label(frame_ra, text="s").grid(row=0, column=5)

    frame_dec = ttk.Frame(root)
    frame_dec.grid(row=1, column=1, padx=10, pady=5)
    dec_degree = tk.StringVar()
    dec_degree_entry = ttk.Entry(frame_dec, width=5, textvari
    dec_degree_entry.grid(row=0, column=0, padx=(0, 5))
    ttk.Label(frame_dec, text="°").grid(row=0, column=1, padx
    dec_minute = tk.StringVar()
    dec_minute_entry = ttk.Entry(frame_dec, width=5, textvari
    dec_minute_entry.grid(row=0, column=2, padx=(0, 5))
    ttk.Label(frame_dec, text="'").grid(row=0, column=3, padx
    dec_second = tk.StringVar()
    dec_second_entry = ttk.Entry(frame_dec, width=5, textvari
    dec_second_entry.grid(row=0, column=4, padx=(0, 5))
    ttk.Label(frame_dec, text="\"").grid(row=0, column=5)

    btn_goto = ttk.Button(root, text="GoTo", command=on_goto)
    btn_goto.grid(row=2, column=0, columnspan=2, padx=10, pad

    separator = ttk.Separator(root, orient='horizontal')
```

```
    separator.grid(row=3, column=0, columnspan=2, sticky='ew'

    btn_track = ttk.Button(root, text="Track", command=on_tra
    btn_track.grid(row=4, column=0, columnspan=2, padx=10, pa

    response_label = tk.Label(root, text="-" * 70 + "\n", fon
    response_label.grid(row=5, column=0, columnspan=2, padx=1

    btn_finish = tk.Button(root, text="Finish Connection", co
    btn_finish.grid(row=6, column=0, columnspan=2, padx=10, p

    thread = threading.Thread(target=message_manager)
    thread.start()

    root.mainloop()
```