Universidade Federal de Ouro Preto

Departamento de Computção

Trabalho Prático de Redes de Computadores

Implementação de Servidor e Cliente FTP

Grupo formado por:

Thalles Felipe Rodrigues de Almeida Santos
Thaís Ferreira de Oliveira Almeida

4 de março de 2025

Sumário

1 Introdução

Este relatório apresenta uma análise detalhada do desenvolvimento de um servidor FTP simples (MyFTP Server) e de um cliente FTP com interface gráfica implementado em Python. O trabalho foi desenvolvido no âmbito da disciplina de Redes de Computadores, na Universidade Federal de Ouro Preto, pelos alunos Thalles Felipe Rodrigues de Almeida Santos e Thaís Ferreira de Oliveira Almeida.

Os códigos implementam funcionalidades básicas de um serviço FTP, abrangendo autenticação de usuários com segurança (utilizando `bcrypt` para hash de senhas), operações de manipulação de arquivos (upload, download, listagem, navegação, criação e remoção de diretórios) e a utilização de concorrência (via `ThreadPoolExecutor`) no servidor. No lado do cliente, a interface gráfica foi desenvolvida com `Tkinter`, permitindo uma interação amigável com o serviço FTP.

2 Estrutura do Projeto

O projeto está dividido em duas partes principais:

- **Servidor FTP (MyFTP Server):** Responsável por gerenciar conexões, autenticar usuários, processar comandos de manipulação de arquivos e gerenciar a comunicação via sockets. Implementa mecanismos de segurança, como bloqueio de usuários após múltiplas tentativas de login malsucedidas e verificação de caminhos para evitar acesso indevido.
- **Cliente FTP com Interface Gráfica:** Desenvolvido com a biblioteca `Tkinter` e seus módulos de `ttk` e `PIL`, possibilita ao usuário realizar operações como login, upload, download e gerenciamento de diretórios através de uma interface intuitiva e responsiva.

3 Descrição do Servidor FTP (MyFTP Server)

3.1 Visão Geral

O servidor FTP é implementado em Python e realiza as seguintes funções principais:

- **Autenticação:** Validação dos usuários utilizando senhas previamente hasheadas com `bcrypt`. Possui mecanismo de bloqueio para evitar tentativas de acesso indevido.
- **Gerenciamento de Arquivos:** Suporta comandos como `put` (upload), `get` (download), `ls` (listagem), `cd` (navegação de diretórios), `mkdir` (criação de diretórios) e `rmdir` (remoção de diretórios).
- **Segurança:** A função `safe_path` garante que as operações de arquivo ocorram apenas dentro de um diretório base definido, prevenindo acesso não autorizado a outras partes do sistema.

- **Concorrência:** Utiliza o `ThreadPoolExecutor` para tratar múltiplos clientes simultaneamente, garantindo que o servidor continue responsivo mesmo com diversas conexões.
- **Logging:** Registra informações sobre conexões, tentativas de login e erros utilizando o módulo `logging`.

3.2 Principais Componentes do Código

- **Configuração Inicial:** Define o diretório base (`server_files`) e configura os parâmetros de segurança, como o número máximo de tentativas de login e o tempo de bloqueio.
- **Funções Auxiliares:**
  - `is_locked_out`: Verifica se um usuário está bloqueado.
  - `record_failed_attempt`: Registra e conta as tentativas de login falhadas.
  - `reset_failed_attempts`: Reseta o contador após um login bem-sucedido.
  - `safe_path`: Realiza a montagem de caminhos garantindo que as operações permaneçam dentro do diretório base.
- **Função `client_handler`:** Gerencia a conexão com cada cliente, interpretando comandos recebidos e executando as operações correspondentes. Esta função implementa o fluxo de autenticação e o processamento de comandos de manipulação de arquivos.
- **Função `start_server`:** Inicializa o servidor, configura o socket para aceitar conexões e utiliza um pool de threads para lidar com múltiplos clientes simultaneamente.

4 Descrição do Cliente FTP com Interface Gráfica

4.1 Visão Geral

O cliente FTP foi implementado com a biblioteca `Tkinter`, proporcionando uma interface gráfica interativa para o usuário. As principais funcionalidades incluem:

- **Tela de Login:** Permite ao usuário inserir o endereço IP do servidor, porta, usuário e senha. Após a autenticação, a interface muda para a tela principal.
- **Interface Principal:** Organizada em dois painéis – uma **sidebar** com botões para operações (upload, download, navegação, criação e remoção de diretórios) e uma área central onde os arquivos são listados utilizando o widget `Treeview`.
- **Operações Assíncronas:** O uso de `threading` garante que operações como upload e download ocorram em segundo plano, evitando que a interface congele durante a transferência de arquivos.
- **Utilização de Ícones:** Para melhorar a experiência do usuário, ícones são carregados (usando `PIL`) e exibidos nos botões.

4.2 Principais Componentes do Código

- **Classe `MyFTPClientGUI`:** Centraliza toda a lógica da interface gráfica e a comunicação com o servidor.
- **Método `login`:** Responsável por estabelecer a conexão via socket com o servidor e enviar o comando de login. Em caso de sucesso, a interface de login é substituída pela tela principal.
- **Métodos de Comando:**
  - `ls_command`: Lista os arquivos do diretório atual.
  - `put_command` e `put_multiple_command`: Gerenciam o upload de um ou múltiplos arquivos.
  - `get_command`: Gerencia o download de um arquivo.
  - `cd_command` e `cd_up_command`: Permitem a navegação entre diretórios.
  - `mkdir_command` e `rmdir_command`: Criam e removem diretórios, respectivamente.
  - `logout`: Finaliza a sessão e retorna à tela de login.
- **Gerenciamento de Threads:** Para cada operação que pode demorar (upload/-download), uma thread separada é iniciada para evitar bloqueios na interface.

5 Detalhamento do Código

5.1 Servidor FTP

- **Segurança e Autenticação:** Ao iniciar, o servidor define um dicionário de usuários com senhas hasheadas. Durante a autenticação, o servidor utiliza funções como `is_locked_out` e `record_failed_attempt` para monitorar e controlar tentativas de login.
- **Processamento de Comandos:** A função `client_handler` interpreta os comandos enviados pelo cliente. Cada comando (por exemplo, `put`, `get`, `ls`, `cd`, etc.) possui um tratamento específico, com verificações de argumentos e mensagens de retorno apropriadas.
- **Concorrência:** Ao utilizar o `ThreadPoolExecutor`, o servidor consegue lidar com múltiplas conexões sem bloquear a execução de novos clientes.
- **Manipulação Segura de Arquivos:** A função `safe_path` garante que operações de leitura e escrita sejam realizadas apenas dentro do diretório predefinido, prevenindo possíveis vulnerabilidades de segurança.

5.2 Cliente FTP

- **Interface Gráfica:** A classe `MyFTPClientGUI` organiza a interface em dois modos: uma tela de login e uma tela principal para operações. A separação de frames facilita a troca de telas após a autenticação.
- **Operações de Arquivo:** Cada operação (upload, download, listagem, etc.) é implementada em métodos específicos que interagem com o servidor através de comandos enviados via socket.
- **Feedback ao Usuário:** O uso de `messagebox` e `simpledialog` permite que o usuário receba respostas imediatas sobre o status das operações, melhorando a experiência de uso.
- **Assincronia:** Para evitar travamentos da interface, operações de longa duração são executadas em threads separadas. Dessa forma, o usuário pode continuar interagindo com a interface enquanto as transferências ocorrem em segundo plano.

6 Conclusão

O projeto apresenta uma implementação funcional de um serviço FTP básico, destacando conceitos essenciais de redes de computadores, como comunicação via sockets, gerenciamento de múltiplas conexões, segurança na autenticação e transferência de arquivos, bem como o desenvolvimento de interfaces gráficas interativas. A separação clara entre as responsabilidades do servidor e do cliente, aliada ao uso de bibliotecas padrão do Python, demonstra uma abordagem prática e eficiente para o desenvolvimento de aplicações de rede.

A Código Fonte do Servidor (MyFTP Server)

```
1  #!/usr/bin/env python3
2  """
3  Servidor FTP simples (MyFTP Server).
4
5  Este script implementa um servidor FTP que realiza opera  es
       b sicas de
6  autentica  o e manipula  o de arquivos (upload, download,
       listagem,
7  navega  o e gerenciamento de diret rios). Utiliza sockets
       para comunica  o
8  com os clientes, bcrypt para hashing de senhas e um
       ThreadPoolExecutor para
9  concorr ncia.
10 """
11
12 import socket
13 import os
```

```
14 import bcrypt
15 import logging
16 import concurrent.futures
17 import time
18 from pathlib import Path
19
20 # Configura  o do logging
21 logging.basicConfig(level=logging.INFO, format="%(asctime)s -
      %(levelname)s - %(message)s")
22
23 # Diret rio base para opera  es de arquivo
24 BASE_DIR: Path = Path("server_files").resolve()
25 if not BASE_DIR.exists():
26     BASE_DIR.mkdir(parents=True)
27
28 # Configura  es de seguran a da conta
29 MAX_FAILED_ATTEMPTS: int = 3       # N mero m ximo de
      tentativas de login falhadas permitidas
30 LOCKOUT_DURATION: int = 60           # Dura  o do bloqueio (em
      segundos)
31
32 # Usu rios pr -hashed utilizando bcrypt
33 users = {
34     "usuario1": bcrypt.hashpw("senha1".encode(),
          bcrypt.gensalt()),
35     "usuario2": bcrypt.hashpw("senha2".encode(),
          bcrypt.gensalt()),
36     "thalles": bcrypt.hashpw("1234".encode(), bcrypt.gensalt()),
37     "thais": bcrypt.hashpw("1234".encode(), bcrypt.gensalt()),
38     "admin": bcrypt.hashpw("admin".encode(), bcrypt.gensalt())
39 }
40
41 # Dicion rio para rastrear tentativas de login falhadas:
      {username: (failed_count, lockout_until)}
42 failed_attempts = {}
43
44
45 def is_locked_out(username: str) -> bool:
46     """
47     Verifica se o usu rio est  bloqueado por m ltiplas
          tentativas falhadas.
48
49     :param username: Nome do usu rio
50     :return: True se o usu rio estiver bloqueado; caso
          contr rio , False.
51     """
52     if username in failed_attempts:
53         _, lockout_until = failed_attempts[username]
54         if time.time() < lockout_until:
```

```
55                 return True
56         return False
57
58
59 def record_failed_attempt(username: str) -> None:
60         """
61         Registra uma tentativa de login falhada e bloqueia a conta,
               se necess rio.
62
63         :param username: Nome do usu rio que teve a tentativa
               falhada.
64         :return: None
65         """
66         current_time = time.time()
67         count = failed_attempts.get(username, (0, current_time))[0]
               + 1
68         if count >= MAX_FAILED_ATTEMPTS:
69             lockout_until = current_time + LOCKOUT_DURATION
70             logging.warning(f"Usu rio {username} bloqueado at
                   {lockout_until}")
71             failed_attempts[username] = (count, lockout_until)
72         else:
73             failed_attempts[username] = (count, current_time)
74
75
76 def reset_failed_attempts(username: str) -> None:
77         """
78         Reseta o contador de tentativas de login falhadas para um
               usu rio.
79
80         :param username: Nome do usu rio.
81         :return: None
82         """
83         if username in failed_attempts:
84             del failed_attempts[username]
85
86
87 def safe_path(current_dir: Path, target: str) -> Path:
88         """
89         Realiza a jun  o segura de um caminho relativo com o
               diret rio atual,
90         garantindo que o caminho resultante permane a dentro do
               BASE_DIR.
91
92         :param current_dir: Diret rio atual.
93         :param target: Caminho ou nome do diret rio/arquivo alvo.
94         :return: Novo caminho resolvido se for v lido; caso
               contr rio, retorna o diret rio atual.
95         """
```

```
96      new_path = (current_dir / target).resolve()
97      if BASE_DIR in new_path.parents or new_path == BASE_DIR:
98          return new_path
99      return current_dir
100
101
102 def client_handler(conn: socket.socket, addr) -> None:
103      """
104      Gerencia a conex o com um cliente: realiza autentica  o,
            processa comandos
105      e executa opera  es de arquivo.
106
107      Comandos suportados:
108        - login <usuario> <senha>
109        - put <nome_arquivo>
110        - get <nome_arquivo>
111        - ls
112        - cd <nome_da_pasta>
113        - cd..
114        - mkdir <nome_da_pasta>
115        - rmdir <nome_da_pasta>
116        - logout
117
118      :param conn: Socket da conex o com o cliente.
119      :param addr: Endere o do cliente.
120      :return: None
121      """
122      logging.info(f"Conex o estabelecida com {addr}")
123      current_dir = BASE_DIR
124      authenticated = False
125      username = None
126
127      # Define timeout para conex es inativas (300 segundos)
128      conn.settimeout(300)
129
130      try:
131          while True:
132              data = conn.recv(1024).decode().strip()
133              if not data:
134                  break
135              logging.info(f"Recebido de {addr}: {data}")
136              parts = data.split()
137
138              # Comando: login
139              if parts[0] == "login":
140                  if len(parts) != 3:
141                      conn.sendall("Erro: Comando login inv lido.
                            Use: login <usuario> <senha>\n".encode())
142                      continue
```

```
143                 _, user, password = parts
144                 if is_locked_out(user):
145                     conn.sendall("Erro: Conta bloqueada devido a
                            m ltiplas tentativas
                            falhadas.\n".encode())
146                     continue
147                 if user in users and
                        bcrypt.checkpw(password.encode(), users[user]):
148                     authenticated = True
149                     username = user
150                     reset_failed_attempts(user)
151                     conn.sendall("Login
                            bem-sucedido!\n".encode())
152                 else:
153                     record_failed_attempt(user)
154                     conn.sendall("Erro: Usu rio ou senha
                            incorretos.\n".encode())
155
156             # Exige autentica  o para os comandos subsequentes
157             elif not authenticated:
158                 conn.sendall("Erro: Voc  deve fazer login
                        primeiro.\n".encode())
159
160             # Comando: put (upload de arquivo)
161             elif parts[0] == "put":
162                 if len(parts) != 2:
163                     conn.sendall("Erro: Comando put inv lido.
                            Use: put <nome_arquivo>\n".encode())
164                     continue
165                 filename = parts[1]
166                 safe_file_path = (current_dir /
                        filename).resolve()
167                 # Verifica se o caminho est  dentro do BASE_DIR
168                 if BASE_DIR not in safe_file_path.parents and
                        safe_file_path != BASE_DIR:
169                     conn.sendall("Erro: Caminho de arquivo
                            inv lido.\n".encode())
170                     continue
171                 conn.sendall("READY".encode())
172                 file_size_str = conn.recv(1024).decode().strip()
173                 try:
174                     file_size = int(file_size_str)
175                 except ValueError:
176                     conn.sendall("Erro: Tamanho de arquivo
                            inv lido.\n".encode())
177                     continue
178                 received_bytes = 0
179                 with open(safe_file_path, "wb") as f:
180                     while received_bytes < file_size:
```

```
                    chunk = conn.recv(4096)
                    if not chunk:
                        break
                    f.write(chunk)
                    received_bytes += len(chunk)
            if received_bytes == file_size:
                conn.sendall("Arquivo recebido com
                    sucesso.\n".encode())
            else:
                conn.sendall("Erro: Transfer ncia
                    incompleta.\n".encode())

        # Comando: get (download de arquivo)
        elif parts[0] == "get":
            if len(parts) != 2:
                conn.sendall("Erro: Comando get inv lido.
                    Use: get <nome_arquivo>\n".encode())
                continue
            filename = parts[1]
            safe_file_path = (current_dir /
                filename).resolve()
            if not (safe_file_path.exists() and (BASE_DIR in
                safe_file_path.parents or safe_file_path ==
                BASE_DIR)):
                conn.sendall("Erro: Arquivo n o
                    encontrado.\n".encode())
                continue
            file_size = os.path.getsize(safe_file_path)
            conn.sendall(str(file_size).encode())
            ack = conn.recv(1024).decode().strip()
            if ack != "READY":
                continue
            with open(safe_file_path, "rb") as f:
                while True:
                    chunk = f.read(4096)
                    if not chunk:
                        break
                    conn.sendall(chunk)
            conn.sendall("Transfer ncia
                conclu da.\n".encode())

        # Comando: ls (listar conte do do diret rio)
        elif parts[0] == "ls":
            try:
                items = os.listdir(current_dir)
                if not items:
                    conn.sendall("Diret rio
                        vazio.".encode())
                else:
```

```
                    lines = []
                    for item in items:
                        full_path = current_dir / item
                        if full_path.is_dir():
                            lines.append(f"{item}:Dir:-")
                        else:
                            size = os.path.getsize(full_path)
                            lines.append(f"{item}:File:{size}")
                    response = "\n".join(lines)
                    conn.sendall(response.encode())
            except Exception as e:
                conn.sendall(f"Erro ao listar diret rio:
                    {str(e)}".encode())

        # Comando: cd (mudar de diret rio)
        elif parts[0] == "cd":
            if len(parts) < 2:
                conn.sendall("Erro: Comando cd inv lido.
                    Use: cd <nome_da_pasta>\n".encode())
                continue
            folder = " ".join(parts[1:])
            new_path = safe_path(current_dir, folder)
            if new_path.exists() and new_path.is_dir():
                current_dir = new_path
                conn.sendall(f"Diret rio alterado para
                    {str(current_dir)}\n".encode())
            else:
                conn.sendall("Erro: Diret rio n o
                    encontrado.\n".encode())

        # Comando: cd.. (voltar um n vel no diret rio)
        elif data == "cd..":
            parent = current_dir.parent
            if BASE_DIR in parent.parents or parent ==
                BASE_DIR:
                current_dir = parent
                conn.sendall(f"Diret rio alterado para
                    {str(current_dir)}\n".encode())
            else:
                conn.sendall("Erro: J  est  no diret rio
                    raiz.\n".encode())

        # Comando: mkdir (criar diret rio)
        elif parts[0] == "mkdir":
            if len(parts) != 2:
                conn.sendall("Erro: Comando mkdir inv lido.
                    Use: mkdir <nome_da_pasta>\n".encode())
                continue
            folder = parts[1]
```

```
262                 new_dir = (current_dir / folder).resolve()
263                 if BASE_DIR not in new_dir.parents and new_dir
                        != BASE_DIR:
264                     conn.sendall("Erro: Caminho
                            inv lido.\n".encode())
265                     continue
266                 try:
267                     os.mkdir(new_dir)
268                     conn.sendall("Diret rio criado com
                            sucesso.\n".encode())
269                 except Exception as e:
270                     conn.sendall(f"Erro ao criar diret rio:
                            {str(e)}\n".encode())
271
272             # Comando: rmdir (remover diret rio)
273             elif parts[0] == "rmdir":
274                 if len(parts) != 2:
275                     conn.sendall("Erro: Comando rmdir inv lido.
                            Use: rmdir <nome_da_pasta>\n".encode())
276                     continue
277                 folder = parts[1]
278                 dir_path = (current_dir / folder).resolve()
279                 if dir_path.exists() and dir_path.is_dir():
280                     try:
281                         os.rmdir(dir_path)
282                         conn.sendall("Diret rio removido com
                                sucesso.\n".encode())
283                     except Exception as e:
284                         conn.sendall(f"Erro ao remover
                                diret rio: {str(e)}\n".encode())
285                 else:
286                     conn.sendall("Erro: Diret rio n o
                            encontrado.\n".encode())
287
288             # Comando: logout (encerrar conex o)
289             elif parts[0] == "logout":
290                 conn.sendall("Logout realizado.\n".encode())
291                 break
292
293             # Comando n o reconhecido
294             else:
295                 conn.sendall("Comando n o
                        reconhecido.\n".encode())
296
297     except socket.timeout:
298         logging.warning(f"Conex o com {addr} expirou por
                timeout.")
299     except Exception as e:
300         logging.error(f"Erro com {addr}: {e}")
```

```
301     finally:
302         conn.close()
303         logging.info(f"Conex o  encerrada com {addr}")
304 
305 
306 def start_server(host: str = "0.0.0.0", port: int = 2121) ->
        None:
307     """
308     Inicializa o servidor MyFTP no host e porta especificados.
309 
310     Cria um socket, configura op  es , associa ao endere o e
            porta, e inicia a
311     escuta de conex es. Utiliza ThreadPoolExecutor para tratar
            m ltiplos clientes
312     de forma concorrente.
313 
314     :param host: Endere o do host (padr o "0.0.0.0" para
            escutar em todas as interfaces).
315     :param port: Porta para escuta (padr o 2121).
316     :return: None
317     """
318     server_socket = socket.socket(socket.AF_INET,
            socket.SOCK_STREAM)
319     # Permite a reutiliza  o do endere o
320     server_socket.setsockopt(socket.SOL_SOCKET,
            socket.SO_REUSEADDR, 1)
321     server_socket.bind((host, port))
322     server_socket.listen(5)
323     server_socket.settimeout(10)
324     logging.info(f"Servidor MyFTP rodando em {host}:{port}")
325 
326     with concurrent.futures.ThreadPoolExecutor(max_workers=10)
            as executor:
327         while True:
328             try:
329                 client_sock, addr = server_socket.accept()
330                 executor.submit(client_handler, client_sock,
                        addr)
331             except socket.timeout:
332                 continue
333             except Exception as e:
334                 logging.error(f"Erro no servidor: {e}")
335 
336 
337 if __name__ == "__main__":
338     start_server()
```

Listing 1: Código Fonte do Servidor FTP

B Código Fonte do Cliente (Interface Gráfica)

```
#!/usr/bin/env python3
"""
Cliente FTP com interface gr fica utilizando Tkinter.

Este script implementa uma interface gr fica para um cliente FTP simples,
permitindo opera  es como login, listagem de arquivos, upload, download,
navega  o de diret rios, cria  o e remo  o de pastas, e logout.
Utiliza sockets para comunica  o com o servidor FTP e threading para
manipula  o de opera  es ass ncronas.
"""

import tkinter as tk
from tkinter import filedialog, messagebox, simpledialog, ttk
import socket
import threading
import os
from PIL import Image, ImageTk


class MyFTPClientGUI:
    """
    Classe que implementa a interface gr fica do cliente FTP.

    Esta classe gerencia a cria  o da interface, conex o com o servidor FTP e
    execu  o de comandos como login, upload, download, navega  o de diret rios,
    cria  o e remo  o de pastas.
    """
    def __init__(self, master: tk.Tk):
        """
        Inicializa a interface gr fica do cliente FTP.

        Configura a janela principal, temas do ttk, frames para login e para a
        interface principal ap s o login, al m de criar bot es e widgets necess rios.

        :param master: Inst ncia da janela Tkinter principal.
        """
        self.master = master
        self.master.title("MyFTP Client")
```

```
39          self.master.geometry("900x600")  # Tamanho aumentado da
               janela
40          self.connection = None
41
42          # Configura  o do tema ttk para um visual moderno
43          style = ttk.Style()
44          style.theme_use('clam')
45
46          # ======= CONFIGURA  O DOS BOT ES (cores e fontes
               padronizadas) =======
47          style.configure(
48              'TButton',
49              font=('Segoe UI', 11),  # Fonte aumentada
50              padding=8,                # Padding aumentado
51              background='#D3D3D3',     # Cinza claro
52              foreground='black'
53          )
54          style.map(
55              'TButton',
56              background=[('active', '#BFBFBF')]  # Cinza mais
                   escuro ao clicar
57          )
58
59          style.configure(
60              'Custom.TButton',
61              background='#D3D3D3',
62              foreground='black',
63              font=('Segoe UI', 12, 'bold'),  # Fonte maior e em
                   negrito para bot es principais
64              padding=10                      # Mais padding para
                   bot es importantes
65          )
66          style.map(
67              'Custom.TButton',
68              background=[('active', '#BFBFBF')]
69          )
70          #
               ================================================================
71
72          style.configure('TLabel', font=('Segoe UI', 11))  #
               Fonte dos r tulos aumentada
73          style.configure('TEntry', font=('Segoe UI', 11))  #
               Fonte dos campos de entrada aumentada
74          style.configure('Treeview', font=('Segoe UI', 11),
               rowheight=30)  # Altura das linhas da Treeview
75          style.configure('Treeview.Heading', font=('Segoe UI',
               11, 'bold'))
76
77          # Configura  o do fundo da janela e dos frames
```

```
78          self.master.configure(bg="white")
79          style.configure("TFrame", background="white")
80          style.configure("TLabel", background="white")
81
82          # Frame container para alternar entre telas (login e
               principal)
83          self.container = ttk.Frame(master)
84          self.container.pack(expand=True, fill='both')
85
86          # ================== FRAME DE LOGIN ==================
87          self.login_frame = ttk.Frame(self.container)
88          self.login_frame.place(relx=0.5, rely=0.5,
               anchor='center')
89
90          # T tulo da tela de login
91          ttk.Label(
92              self.login_frame,
93              text="MyFTP Login",
94              font=('Segoe UI', 18, 'bold')
95          ).grid(row=0, column=0, columnspan=2, pady=20)
96
97          # Campo para informar o IP do servidor
98          ttk.Label(
99              self.login_frame,
100             text="Servidor IP:",
101             font=('Segoe UI', 12)
102         ).grid(row=1, column=0, sticky='e', padx=10, pady=8)
103         self.server_ip_entry = ttk.Entry(self.login_frame,
               font=('Segoe UI', 12), width=25)
104         self.server_ip_entry.grid(row=1, column=1, padx=10,
               pady=8)
105         self.server_ip_entry.insert(0, "127.0.0.1")
106
107         # Campo para informar a porta
108         ttk.Label(
109             self.login_frame,
110             text="Porta:",
111             font=('Segoe UI', 12)
112         ).grid(row=2, column=0, sticky='e', padx=10, pady=8)
113         self.port_entry = ttk.Entry(self.login_frame,
               font=('Segoe UI', 12), width=25)
114         self.port_entry.grid(row=2, column=1, padx=10, pady=8)
115         self.port_entry.insert(0, "2121")
116
117         # Campo para informar o usu rio
118         ttk.Label(
119             self.login_frame,
120             text="Usu rio:",
121             font=('Segoe UI', 12)
```

```
122         ).grid(row=3, column=0, sticky='e', padx=10, pady=8)
123         self.user_entry = ttk.Entry(self.login_frame,
               font=('Segoe UI', 12), width=25)
124         self.user_entry.grid(row=3, column=1, padx=10, pady=8)
125
126         # Campo para informar a senha (oculta)
127         ttk.Label(
128             self.login_frame,
129             text="Senha:",
130             font=('Segoe UI', 12)
131         ).grid(row=4, column=0, sticky='e', padx=10, pady=8)
132         self.pass_entry = ttk.Entry(self.login_frame, show="*",
               font=('Segoe UI', 12), width=25)
133         self.pass_entry.grid(row=4, column=1, padx=10, pady=8)
134
135         # Frame para centralizar o bot o de login
136         login_btn_frame = ttk.Frame(self.login_frame)
137         login_btn_frame.grid(row=5, column=0, columnspan=2,
               pady=20)
138         self.login_button = ttk.Button(
139             login_btn_frame,
140             text="Login",
141             command=self.login,
142             style='Custom.TButton',
143             width=15
144         )
145         self.login_button.pack(pady=5)
146
147         # R tulo de rodap  na tela de login
148         self.login_footer_label = ttk.Label(
149             self.master,
150             text="Desenvolvido por Tha s e Thalles",
151             font=('Segoe UI', 9)
152         )
153         self.login_footer_label.place(relx=0.5, rely=0.98,
               anchor='s')
154         # ========================================================
155
156         # ================== FRAME PRINCIPAL ==================
157         self.main_frame = ttk.Frame(self.container)
158
159         # Sidebar para os bot es de comando
160         self.sidebar = ttk.Frame(self.main_frame, width=200)
161         self.sidebar.grid(row=0, column=0, sticky='ns',
               padx=(10, 5), pady=10)
162         self.sidebar.pack_propagate(False)  # Impede que a
               sidebar encolha
163
164         # Frame para exibi  o dos arquivos (Treeview)
```

```
165         self.file_frame = ttk.Frame(self.main_frame, padding=10)
166         self.file_frame.grid(row=0, column=1, sticky='nsew',
               padx=(5, 10), pady=10)
167 
168         # Configura  o de responsividade da interface principal
169         self.main_frame.columnconfigure(1, weight=1)
170         self.main_frame.rowconfigure(0, weight=1)
171 
172         # Treeview com barra de rolagem para listar arquivos
173         self.scrollbar = ttk.Scrollbar(self.file_frame)
174         self.scrollbar.pack(side='right', fill='y')
175         self.ls_tree = ttk.Treeview(
176             self.file_frame,
177             columns=("Name", "Type", "Size"),
178             show="headings",
179             yscrollcommand=self.scrollbar.set
180         )
181         self.ls_tree.heading("Name", text="Nome")
182         self.ls_tree.heading("Type", text="Tipo")
183         self.ls_tree.heading("Size", text="Tamanho")
184         self.ls_tree.column("Name", width=250)
185         self.ls_tree.column("Type", width=100)
186         self.ls_tree.column("Size", width=120)
187         self.ls_tree.pack(expand=True, fill='both')
188         self.scrollbar.config(command=self.ls_tree.yview)
189 
190         # Carregamento dos  cones  para os bot es da sidebar
191         self.upload_icon =
               ImageTk.PhotoImage(Image.open("upload.png").resize((24,
               24)))
192         self.download_icon =
               ImageTk.PhotoImage(Image.open("download.png").resize((24,
               24)))
193         self.folder_icon =
               ImageTk.PhotoImage(Image.open("folder.png").resize((24,
               24)))
194         self.back_icon =
               ImageTk.PhotoImage(Image.open("back.png").resize((24,
               24)))
195         self.create_icon =
               ImageTk.PhotoImage(Image.open("create.png").resize((24,
               24)))
196         self.delete_icon =
               ImageTk.PhotoImage(Image.open("delete.png").resize((24,
               24)))
197         self.logout_icon =
               ImageTk.PhotoImage(Image.open("logout.png").resize((24,
               24)))
```

```
198         self.reload_icon =
               ImageTk.PhotoImage(Image.open("reload.png").resize((24,
               24)))
199
200         # Defini  o dos bot es com suas fun  es e  cones
201         btn_options = [
202             ("Atualizar Lista (ls)", self.ls_command,
                   self.reload_icon),
203             ("Upload (put)", self.put_command, self.upload_icon),
204             ("Upload M ltiplo", self.put_multiple_command,
                   self.upload_icon),
205             ("Download (get)", self.get_command,
                   self.download_icon),
206             ("Mudar Diret rio (cd)", self.cd_command,
                   self.folder_icon),
207             ("Voltar Diret rio (cd..)", self.cd_up_command,
                   self.back_icon),
208             ("Criar Pasta (mkdir)", self.mkdir_command,
                   self.create_icon),
209             ("Remover Pasta (rmdir)", self.rmdir_command,
                   self.delete_icon),
210             ("Logout", self.logout, self.logout_icon)
211         ]
212         for i, (text, cmd, icon) in enumerate(btn_options):
213             btn = ttk.Button(self.sidebar, text=text,
                       image=icon, compound='left', command=cmd, width=20)
214             btn.pack(fill='x', padx=5, pady=6)
215         # ========================================================
216
217     def login(self) -> None:
218         """
219         Realiza o login no servidor FTP.
220
221         Obt m as informa  es de conex o (IP, porta, usu rio
               e senha) dos campos de entrada,
222         estabelece a conex o via socket, envia o comando de
               login e, se bem-sucedido,
223         troca a tela de login pela tela principal.
224
225         :return: None
226         """
227         server_ip = self.server_ip_entry.get()
228         port = int(self.port_entry.get())
229         user = self.user_entry.get()
230         pwd = self.pass_entry.get()
231
232         try:
233             self.connection = socket.socket(socket.AF_INET,
                       socket.SOCK_STREAM)
```

```
234             self.connection.connect((server_ip, port))
235             # Conex o estabelecida (mensagem de status omitida)
236         except Exception as e:
237             messagebox.showerror("Erro", f"N o foi poss vel
                    conectar ao servidor: {e}")
238             return
239 
240         login_command = f"login {user} {pwd}"
241         self.connection.sendall(login_command.encode())
242         response = self.connection.recv(1024).decode()
243         if "bem-sucedido" in response:
244             messagebox.showinfo("Login", "Login realizado com
                    sucesso!")
245             self.login_frame.destroy()          # Remove a tela
                    de login
246             self.login_footer_label.destroy()     # Remove o
                    rodap  do login
247             self.main_frame.pack(expand=True, fill='both')
248             self.ls_command()  # Atualiza a lista de arquivos
249         else:
250             messagebox.showerror("Login", response)
251 
252     def ls_command(self) -> None:
253         """
254         Lista os arquivos no diret rio atual do servidor FTP.
255 
256         Envia o comando 'ls' ao servidor, limpa a Treeview e
                popula com os dados
257         retornados.
258 
259         :return: None
260         """
261         try:
262             self.connection.sendall("ls".encode())
263             response = self.connection.recv(4096).decode()
264             # Limpa a Treeview antes de inserir os novos dados
265             for item in self.ls_tree.get_children():
266                 self.ls_tree.delete(item)
267             # Processa cada linha da resposta e insere na
                    Treeview
268             for line in response.split("\n"):
269                 if line:
270                     name, type_, size = line.split(":")
271                     self.ls_tree.insert("", "end", values=(name,
                            type_, size))
272         except Exception as e:
273             messagebox.showerror("Erro", str(e))
274 
275     def disable_buttons(self) -> None:
```

```
276             """
277             Desabilita todos os bot es na sidebar durante
                    opera  es  cr ticas.
278
279             Isto evita que o usu rio execute m ltiplas a  es
                    simultaneamente.
280             """
281             for child in self.sidebar.winfo_children():
282                 if isinstance(child, ttk.Button):
283                     child.config(state='disabled')
284
285         def enable_buttons(self) -> None:
286             """
287             Habilita todos os bot es na sidebar ap s a conclus o
                    de uma opera  o.
288             """
289             for child in self.sidebar.winfo_children():
290                 if isinstance(child, ttk.Button):
291                     child.config(state='normal')
292
293         def put_file_thread(self, filepath: str) -> None:
294             """
295             Executa o upload de um  nico  arquivo em uma thread
                    separada.
296
297             Desabilita os bot es, envia o comando de upload,
                    transmite o arquivo em
298             blocos e, ao final, reativa os bot es e atualiza a
                    lista de arquivos.
299
300             :param filepath: Caminho completo do arquivo a ser
                    enviado.
301             :return: None
302             """
303             self.disable_buttons()
304             try:
305                 filename = os.path.basename(filepath)
306                 command = f"put {filename}"
307                 self.connection.sendall(command.encode())
308                 response =
                        self.connection.recv(1024).decode().strip()
309                 if response != "READY":
310                     messagebox.showerror("Erro", f"Servidor
                            respondeu: {response}")
311                     return
312                 file_size = os.path.getsize(filepath)
313                 self.connection.sendall(str(file_size).encode())
314                 sent_bytes = 0
315                 with open(filepath, "rb") as f:
```

```
316                 while sent_bytes < file_size:
317                     chunk = f.read(4096)
318                     if not chunk:
319                         break
320                     self.connection.sendall(chunk)
321                     sent_bytes += len(chunk)
322             final_response = self.connection.recv(1024).decode()
323             messagebox.showinfo("Upload", final_response)
324             self.ls_command()  # Atualiza a lista de arquivos
325         except Exception as e:
326             messagebox.showerror("Erro", str(e))
327         finally:
328             self.enable_buttons()
329 
330     def put_command(self) -> None:
331         """
332         Inicia o upload de um arquivo  nico .
333 
334         Abre uma janela para sele  o do arquivo e inicia a
               thread de upload.
335 
336         :return: None
337         """
338         filepath = filedialog.askopenfilename()
339         if not filepath:
340             return
341         threading.Thread(target=self.put_file_thread,
               args=(filepath,), daemon=True).start()
342 
343     def put_multiple_command(self) -> None:
344         """
345         Inicia o upload de m ltiplos arquivos.
346 
347         Abre uma janela para sele  o de m ltiplos arquivos e
               inicia a thread que
348         gerencia o upload sequencial.
349 
350         :return: None
351         """
352         filepaths = filedialog.askopenfilenames()
353         if not filepaths:
354             return
355         threading.Thread(target=self.put_multiple_files,
               args=(filepaths,), daemon=True).start()
356 
357     def put_multiple_files(self, filepaths: tuple) -> None:
358         """
359         Realiza o upload sequencial de m ltiplos arquivos.
360 
```

```
361         Para cada arquivo selecionado, chama a fun   o de
               upload individual.
362 
363         :param filepaths: Tupla contendo os caminhos completos
               dos arquivos.
364         :return: None
365         """
366         for fp in filepaths:
367             self.put_file_thread(fp)
368 
369     def get_file_thread(self, filename: str) -> None:
370         """
371         Executa o download de um arquivo em uma thread separada.
372 
373         Envia o comando 'get', recebe o tamanho do arquivo,
               solicita confirma  o ,
374         recebe o arquivo em blocos e salva no caminho definido
               pelo usu rio.
375 
376         :param filename: Nome do arquivo a ser baixado.
377         :return: None
378         """
379         self.disable_buttons()
380         try:
381             command = f"get {filename}"
382             self.connection.sendall(command.encode())
383             response =
                   self.connection.recv(1024).decode().strip()
384             if response.startswith("Erro"):
385                 messagebox.showerror("Erro", response)
386                 return
387             try:
388                 file_size = int(response)
389             except ValueError:
390                 messagebox.showerror("Erro", "Tamanho de arquivo
                       inv lido.")
391                 return
392             self.connection.sendall("READY".encode())
393             save_path =
                   filedialog.asksaveasfilename(initialfile=filename)
394             if not save_path:
395                 return
396             received_bytes = 0
397             with open(save_path, "wb") as f:
398                 while received_bytes < file_size:
399                     chunk = self.connection.recv(4096)
400                     if not chunk:
401                         break
402                     f.write(chunk)
```

```
403                     received_bytes += len(chunk)
404             messagebox.showinfo("Download", "Arquivo baixado com
                    sucesso!")
405             self.ls_command()  # Atualiza a lista de arquivos
406         except Exception as e:
407             messagebox.showerror("Erro", str(e))
408         finally:
409             self.enable_buttons()
410 
411     def get_command(self) -> None:
412         """
413         Inicia o download de um arquivo.
414 
415         Solicita ao usu rio o nome do arquivo e inicia a thread
                de download.
416 
417         :return: None
418         """
419         filename = simpledialog.askstring("Download", "Digite o
                nome do arquivo:")
420         if not filename:
421             return
422         threading.Thread(target=self.get_file_thread,
                args=(filename,), daemon=True).start()
423 
424     def cd_command(self) -> None:
425         """
426         Muda para um diret rio especificado no servidor.
427 
428         Solicita o nome do diret rio e envia o comando
                correspondente.
429 
430         :return: None
431         """
432         folder = simpledialog.askstring("cd", "Digite o nome do
                diret rio:")
433         if not folder:
434             return
435         command = f"cd {folder}"
436         self.connection.sendall(command.encode())
437         response = self.connection.recv(1024).decode()
438         messagebox.showinfo("cd", response)
439         self.ls_command()
440 
441     def cd_up_command(self) -> None:
442         """
443         Volta um n vel no diret rio atual do servidor.
444 
```

```
445         Envia o comando para subir um n vel e atualiza a lista
                de arquivos.
446 
447         :return: None
448         """
449         command = "cd.."
450         self.connection.sendall(command.encode())
451         response = self.connection.recv(1024).decode()
452         messagebox.showinfo("cd..", response)
453         self.ls_command()
454 
455     def mkdir_command(self) -> None:
456         """
457         Cria um novo diret rio no servidor.
458 
459         Solicita o nome da nova pasta, envia o comando de
                cria   o e atualiza
460         a lista de arquivos.
461 
462         :return: None
463         """
464         folder = simpledialog.askstring("mkdir", "Digite o nome
                da nova pasta:")
465         if not folder:
466             return
467         command = f"mkdir {folder}"
468         self.connection.sendall(command.encode())
469         response = self.connection.recv(1024).decode()
470         messagebox.showinfo("mkdir", response)
471         self.ls_command()
472 
473     def rmdir_command(self) -> None:
474         """
475         Remove um diret rio do servidor.
476 
477         Solicita o nome da pasta a ser removida, envia o comando
                de remo   o e
478         atualiza a lista de arquivos.
479 
480         :return: None
481         """
482         folder = simpledialog.askstring("rmdir", "Digite o nome
                da pasta a ser removida:")
483         if not folder:
484             return
485         command = f"rmdir {folder}"
486         self.connection.sendall(command.encode())
487         response = self.connection.recv(1024).decode()
488         messagebox.showinfo("rmdir", response)
```

```
489             self.ls_command()
490
491         def logout(self) -> None:
492             """
493             Realiza o logout do servidor FTP e reinicia a interface.
494
495             Envia o comando de logout, fecha a conex o e recria a
                    interface inicial.
496
497             :return: None
498             """
499             try:
500                 self.connection.sendall("logout".encode())
501                 response = self.connection.recv(1024).decode()
502                 messagebox.showinfo("Logout", response)
503                 self.connection.close()
504             except Exception as e:
505                 messagebox.showerror("Erro", str(e))
506             finally:
507                 # Remove todos os widgets e reinicia a interface
508                 self.container.destroy()
509                 self.__init__(self.master)
510
511
512 if __name__ == "__main__":
513     root = tk.Tk()
514     app = MyFTPClientGUI(root)
515     root.mainloop()
```

Listing 2: Código Fonte do Cliente FTP