UFCD – 5119 PROJETO 3
TFTPY CLIENTE

Equipa Técnica – Carlos Oliveira e Flávio Matos

9 DE JULHO DE 2023

INSTITUTO DE EMPREGO E FORMAÇÃO PROFISSIONAL DE ALCÂNTARA

Índice

Introdução

Este relatório profissional fornece uma visão geral e análise da implementação de um cliente TFTP (Trivial File Transfer Protocol) em Python. O cliente TFTP permite a recuperação e transferência de arquivos entre um cliente e um servidor TFTP. O relatório discute o design, funcionalidade e componentes-chave do cliente, juntamente com quaisquer modificações feitas no código original.

O relatório começa por descrever o propósito e a importância do TFTP como um protocolo de transferência de arquivos leve amplamente utilizado em aplicações de redes. Destaca o papel do cliente na iniciação das transferências de arquivos e explora as operações de protocolo subjacentes, como Solicitação de Leitura (RRQ), Solicitação de Escrita (WRQ), Dados (DAT), Confirmação (ACK) e Erro (ERR).

Em seguida, o relatório adentra nos detalhes da implementação do cliente TFTP, focando nas principais funções e suas funcionalidades. Aborda funções como get_file() e put_file(), que lidam com a recuperação e transferência de arquivos, respetivamente. O relatório explica como essas funções estabelecem comunicação com o servidor, lidam com pacotes de dados e gerenciam confirmações e erros.

Além disso, o relatório destaca os mecanismos de tratamento de erros incorporados ao cliente para garantir robustez e confiabilidade. Discute as exceções personalizadas, incluindo Err, ProtocolError e NetworkError, que permitem que o cliente lide com vários cenários de erro de forma adequada. O relatório examina também os códigos e mensagens de erro definidos na implementação e sua importância.

O relatório explora a interface de linha de comando (CLI) fornecida pelo cliente, permitindo que os usuários interajam com o cliente TFTP e especifiquem a operação desejada (obter ou colocar), endereço do servidor, arquivo de origem e arquivo de destino. Descreve os argumentos de linha de comando e seu uso, demonstrando como invocar o cliente com os argumentos apropriados.

Adicionalmente, o relatório aborda quaisquer modificações feitas no código original para aprimorar a funcionalidade e resolver problemas identificados. Destaca mudanças específicas feitas para lidar com tempos limite e aprimorar o tratamento de erros durante a comunicação em rede. O relatório discute os desafios enfrentados durante o processo de modificação e as soluções correspondentes implementadas.

Por fim, o relatório conclui resumindo as conquistas e capacidades da implementação do cliente TFTP. Enfatiza a capacidade do cliente de estabelecer comunicação com um servidor TFTP, recuperar ou transferir arquivos de forma confiável e lidar com erros de forma eficaz. O relatório também menciona áreas potenciais para melhorias ou extensões adicionais, como a implementação de códigos de erro adicionais ou suporte a diferentes modos de transferência.

Em conclusão, este relatório profissional fornece uma visão abrangente da implementação de um cliente TFTP em Python. Destaca o design, funcionalidade, mecanismos de tratamento de erros e interface de linha de comando do cliente. O relatório serve como um recurso valioso para entender o protocolo TFTP e sua aplicação prática em transferências de arquivos.

Vantagens e desvantagens

Utilizar este cliente TFTP apresenta várias vantagens e desvantagens. Vamos analisá-las em detalhes:

Vantagens

1. Simplicidade: O protocolo TFTP é simples e possui uma implementação básica, o que torna o cliente TFTP igualmente simples de usar. Isso é vantajoso para usuários que desejam uma solução simples e direta para transferência de arquivos.

2. Baixo overhead: O TFTP é projetado para ter um overhead mínimo, o que significa que consome menos recursos de rede em comparação com outros protocolos de transferência de arquivos mais complexos. Isso é especialmente útil em redes com largura de banda limitada ou com alta latência.

3. Implementação amplamente suportada: O TFTP é um protocolo bem estabelecido e amplamente suportado por diferentes sistemas e dispositivos. Isso significa que o cliente TFTP pode ser executado em uma variedade de plataformas e ser usado para transferir arquivos de e para diferentes tipos de servidores TFTP.

4. Operação em modo interativo: O cliente TFTP inclui um modo interativo que permite aos usuários executar comandos diretamente a partir de uma interface de linha de comando. Isso oferece flexibilidade e conveniência para a transferência de arquivos, especialmente em ambientes onde o uso de uma interface gráfica não é possível ou prático.

5. Transferência de arquivos rápidas: O TFTP é otimizado para transferências rápidas de arquivos, o que é útil quando a velocidade de transferência é uma prioridade. Isso pode ser especialmente vantajoso em cenários onde grandes volumes de dados precisam ser transferidos em um curto período de tempo.

Desvantagens

1. Falta de segurança: O protocolo TFTP não inclui recursos avançados de segurança, como autenticação ou criptografia de dados. Isso significa que as transferências de arquivos realizadas pelo cliente TFTP podem estar vulneráveis a ataques ou intercetação de dados por terceiros mal-intencionados.

2. Limitações de tamanho de arquivo: O TFTP possui uma restrição no tamanho máximo dos arquivos que podem ser transferidos, geralmente limitado a 32 ou 64 megabytes. Isso pode ser problemático quando há necessidade de transferir arquivos grandes.

3. Ausência de controle de integridade: O TFTP não oferece um mecanismo embutido para verificar a integridade dos dados transferidos. Isso significa que não há garantia de que os arquivos transferidos sejam totalmente consistentes ou livres de erros.

4. Dependência de uma conexão estável: O TFTP requer uma conexão de rede estável para realizar transferências de arquivos bem-sucedidas. Se a conexão for interrompida durante a transferência, o cliente TFTP não possui mecanismos de retomada de transferência automática.

5. Limitações de funcionalidades: O TFTP é um protocolo bastante básico e possui funcionalidades limitadas em comparação com outros protocolos de transferência de arquivos mais avançados, como FTP ou SFTP. Recursos como listagem de diretórios, controle de permissões ou suporte a transferências em modo passivo não estão presentes no cliente TFTP.

Em conclusão, utilizar este cliente TFTP pode ser vantajoso em situações em que a simplicidade, a eficiência de transferência e a ampla compatibilidade são prioridades. No entanto, é importante considerar as desvantagens, como a falta de segurança, as limitações de tamanho de arquivo e a ausência de controle de integridade, antes de escolher o TFTP como solução de transferência de arquivos em um ambiente profissional.

Desenvolvimento

Vamos analisar o código linha por linha

import os: Importa o módulo os para interagir com o sistema operativo.

import sys: Importa o módulo sys para aceder a argumentos da linha de comandos e outros recursos relacionados com o sistema.

import argparse: Importa o módulo argparse para processar argumentos da linha de comandos.

import ipaddress: Importa o módulo ipaddress para validar endereços IP.

import re: Importa o módulo re para fazer correspondência de padrões em strings.

import struct: Importa o módulo struct para realizar conversões entre valores Python e objetos binários.

from socket import socket, herror, gaierror, gethostbyaddr, gethostbyname_ex, AF_INET, SOCK_DGRAM: Importa classes e constantes relacionadas com operações de socket necessárias para comunicação em rede.

A partir da linha 12, temos a definição de constantes que serão utilizadas ao longo do código. Estas constantes incluem o tamanho máximo dos dados (MAX_DATA_LEN), o número máximo de bloco (MAX_BLOCK_NUM), o tempo máximo de inatividade (INACTIVITY_TIMEOUT), o modo padrão de transferência de ficheiros (DEFAULT_MODE), o tamanho do buffer padrão (DEFAULT_BUFFER_SIZE) e a porta padrão (DEFAULT_PORT) do servidor TFTP.

A partir da linha 27, temos a definição de constantes numéricas que representam os códigos de operação do protocolo TFTP. Estes códigos incluem RRQ (Read Request), WRQ (Write Request), DAT (Data), ACK (Acknowledgment) e ERR (Error). Além disso, temos o código LIST (List) adicionado para suportar a listagem de ficheiros no cliente.

A partir da linha 37, temos a definição de uma tabela de mensagens de erro (ERROR_MESSAGES) que associa códigos de erro a mensagens descritivas.

A partir da linha 40, temos a definição do tipo de dados INET4Address, que é uma tupla contendo um endereço IP (string) e uma porta (inteiro).

A partir da linha 43, temos a definição da função get_file que realiza a operação de transferência de um ficheiro do servidor TFTP. Esta função recebe o endereço do servidor (server_addr), o nome do ficheiro remoto (file_name) e o nome do ficheiro de destino (dest_file).

A função get_file cria um socket UDP (linha 44) e define um tempo limite de inatividade para o socket (linha 45). Em seguida, empacota um pedido de leitura (RRQ) contendo o nome do ficheiro (linha 46) e envia-o para o servidor (linha 47).

A função entra num ciclo que recebe pacotes do servidor (linha 49) e verifica o código de operação do pacote recebido (linha 51). Se o código de operação for DAT (linha 53), o pacote contém dados do ficheiro e o número de bloco correspondente. O número de bloco é verificado para garantir a ordem correta (linhas 54-56). Os dados do pacote são escritos no ficheiro de destino (linha 58), e um pacote de confirmação (ACK) é enviado de volta para o servidor (linha 61). Se o tamanho dos dados do pacote for menor que o tamanho máximo permitido (MAX_DATA_LEN), isso significa que o ficheiro foi totalmente transferido e o ciclo é interrompido (linha 63).

Se o código de operação for ERR (linha 64), o pacote contém um código de erro e uma mensagem de erro. Uma exceção (Err) é lançada com o código de erro e a mensagem de erro recebidos (linha 66).

Se o código de operação não for DAT nem ERR, isso indica um erro de protocolo, e uma exceção (ProtocolError) é lançada (linha 68).

A partir da linha 71, temos a definição da função put_file que realiza a operação de envio de um ficheiro para o servidor TFTP. Esta função recebe o endereço do servidor (server_addr), o nome do ficheiro local (file_name) e o nome do ficheiro remoto (dest_file).

A função put_file cria um socket UDP (linha 72) e define um tempo limite de inatividade para o socket (linha 73). Em seguida, empacota um pedido de escrita (WRQ) contendo o nome do ficheiro (linha 74) e envia-o para o servidor (linha 75).

A função entra num ciclo que lê dados do ficheiro local em blocos (linha 77) e empacota os dados em pacotes (DAT) contendo o número de bloco correspondente (linha 79). Os pacotes são enviados para o servidor (linha 81). Em seguida, o ciclo aguarda a confirmação (ACK) do servidor (linha 84). Se o número de bloco do pacote de confirmação corresponder ao próximo número de bloco esperado (linha 86), o próximo número de bloco é incrementado (linha 87). Se o número de bloco não corresponder ao esperado, uma exceção (ProtocolError) é lançada (linha 89).

Se o código de operação do pacote recebido for ERR (linha 90), uma exceção (Err) é lançada com o código de erro e a mensagem de erro recebidos (linha 92).

Se o código de operação não for ACK nem ERR, isso indica um erro de protocolo, e uma exceção (ProtocolError) é lançada (linha 94).

A partir da linha 97, temos a definição da função list_client_directories que lista os ficheiros num diretório do cliente. Esta função recebe um diretório como argumento (opcional) e usa o módulo os para listar os ficheiros no diretório especificado (linha 101).

A partir da linha 104, temos a definição da função pack_list que empacota um pedido de listagem de ficheiros (LIST) contendo o nome do diretório. O nome do diretório é codificado em bytes usando UTF-8 e adicionado a uma estrutura de empacotamento específica (linha 108).

A partir da linha 112, temos a definição da função unpack_list que extrai o nome do diretório de um pacote de resposta de listagem (LIST). A função descodifica o nome do diretório a partir do pacote recebido (linha 113).

A partir da linha 116, temos a definição da função print_help que exibe uma mensagem de ajuda com os comandos disponíveis.

A partir da linha 122, temos a definição da função quit_program que termina o programa.

A partir da linha 125, temos a definição da função print_menu que exibe o menu de comandos disponíveis no modo interativo.

A partir da linha 134, temos a definição da função run_interactive_mode que implementa o modo interativo do cliente TFTP. A função exibe o menu inicial (linha 136) e entra num ciclo para receber comandos do utilizador (linha 139). Dependendo do comando fornecido, as funções apropriadas são chamadas para executar as operações correspondentes, como listagem de ficheiros, download ou upload. O ciclo continua até que o comando "quit" seja digitado.

A partir da linha 161, temos a definição das funções de empacotamento e desempacotamento de pacotes TFTP. Estas funções são usadas para converter estruturas de dados em sequências de bytes compatíveis com o protocolo TFTP e vice-versa.

A partir da linha 193, temos a definição da classe Err que representa uma exceção personalizada para erros do cliente TFTP. A classe armazena o código de erro e a mensagem de erro associada.

A partir da linha 199, temos a definição da classe ProtocolError que representa uma exceção para erros de protocolo do cliente TFTP.

A partir da linha 203, temos a definição da classe NetworkError que representa uma exceção para erros de rede do cliente TFTP.

A partir da linha 207, temos a definição da função validate_ipv4_address que verifica se um endereço IP é válido usando o módulo ipaddress.

A partir da linha 215, temos a definição da função resolve_server_address que recebe um endereço ou nome de anfitrião do servidor e retorna o endereço IP e a porta correspondente. A função verifica se o endereço fornecido é um endereço IP válido. Se não for, a função tenta resolver o nome de anfitrião para obter os endereços IP associados. Se for encontrado um endereço IP válido, este é retornado juntamente com a porta padrão do servidor TFTP. Se não for encontrado um endereço IP válido, é lançada uma exceção.

A partir da linha 233, temos a definição da função parse_arguments que usa o módulo argparse para analisar os argumentos da linha de comandos. A função especifica os argumentos esperados, como a operação (get ou put), a porta do servidor, o endereço do servidor, o ficheiro de origem e o ficheiro de destino. Os argumentos são retornados como um objeto argparse.Namespace.

A partir da linha 249, temos a definição da função main que é o ponto principal de entrada do programa. A função verifica a operação especificada na linha de comandos e chama as funções apropriadas para executar as operações de download ou upload. Se a operação for inválida, é lançada uma exceção.

A partir da linha 265, temos a verificação se o código está a ser executado diretamente como um programa. Se for o caso, o código analisa os argumentos da linha de comandos, verifica se o endereço do servidor foi fornecido e executa o modo interativo chamando a função run_interactive_mode.

Como criar conta e adicionar ficheiros ao Github

1.Criar uma conta no GitHub

- Acesse o site do GitHub em https://github.com;
- Clique em "Sign up" (Registrar-se) no canto superior direito;
- Preencha o formulário com seu nome de usuário, endereço de e-mail e senha;
- Selecione um plano (geralmente o plano gratuito é suficiente para a maioria dos usuários);
- Complete o processo de registo seguindo as instruções fornecidas;

2.Instalar o Git no Linux Mint

- Abra o terminal no Mint;
- Execute o seguinte comando para instalar o Git:
  sudo apt-get update
  sudo apt-get install git

3.Configurar o Git

- Configure seu nome de usuário executando o seguinte comando no terminal, substituindo "Seu Nome" pelo seu nome:
  git config --global user.name "Seu Nome"

- Configure seu endereço de e-mail executando o seguinte comando no terminal, substituindo "seuemail@example.com" pelo seu endereço de e-mail:
  git config --global user.email seuemail@example.com

4.Criar um novo repositório

- Acesse o GitHub em https://github.com e faça login na sua conta;
- Clique no botão "New" (Novo) no canto superior esquerdo da página inicial;
- Digite um nome para o repositório e uma descrição (opcional);
- Escolha as configurações desejadas para o repositório;
- Clique em "Create repository" (Criar repositório) para criar o novo repositório;

5.Transferir arquivos para o repositório usando a linha de comando

- No terminal, navegue até a pasta onde estão os arquivos que você deseja transferir para o repositório;
- Inicialize o repositório Git local executando o seguinte comando:

  git init

- Adicione os arquivos ao repositório executando o seguinte comando:

  git add .

- Comite os arquivos ao repositório executando o seguinte comando:
  git commit -m "Mensagem de commit"
- Adicione o URL do repositório remoto ao repositório local executando o seguinte comando:
  git remote add origin URL-do-repositório-remoto
- Envie os arquivos para o repositório remoto executando o seguinte comando:
  git push -u origin branch-name
- Substitua "branch-name" pelo nome da branch que você deseja usar:
  git branch -M “nome”

Lembre-se de substituir as informações específicas, como nome de usuário, endereço de e-mail, URL do repositório e nome da branch, pelos seus próprios dados.

Protocolo UDP-TCP

O Protocolo UDP (User Datagram Protocol) é um protocolo de transporte utilizado na Internet para o envio de datagramas sem conexão. Ao contrário do TCP (Transmission Control Protocol), que oferece um serviço orientado à conexão e fiável, o UDP é um protocolo simples e não orientado à conexão.

A principal diferença entre o UDP e o TCP é a falta de um mecanismo de controlo de fluxo, controlo de congestionamento e garantia de entrega de pacotes no UDP. Enquanto o TCP estabelece uma ligação entre o emissor e o recetor, com um aperto de mão inicial, sequenciação de pacotes e confirmações de receção, o UDP não possui essas funcionalidades.

Em vez disso, o UDP oferece um serviço de entrega de pacotes não fiável, o que significa que os pacotes podem ser perdidos, duplicados ou entregues fora de ordem. O UDP é geralmente utilizado em aplicações que não exigem fiabilidade absoluta, mas priorizam a velocidade e a baixa latência, como jogos online, streaming de vídeo e aplicações de VoIP (Voz sobre IP).

Outra diferença notável é que o UDP não possui um conceito de bufferização de pacotes como o TCP, o que significa que os pacotes podem ser enviados em rajadas rápidas, sem pausas, o que pode levar à perda de pacotes em caso de congestionamento na rede. Além disso, o UDP não garante a entrega ordenada dos pacotes, o que pode ser problemático para algumas aplicações sensíveis à ordem, como transferências de ficheiros.

Embora o UDP não ofereça as funcionalidades avançadas de controlo e fiabilidade do TCP, possui uma vantagem em termos de eficiência e velocidade. O UDP possui um cabeçalho menor em comparação com o TCP, o que reduz a sobrecarga na rede e pode ser benéfico em ambientes de rede de alta velocidade ou com restrições de largura de banda.

Em resumo, o UDP é um protocolo de transporte simples e não orientado à conexão, que oferece uma entrega de pacotes não fiável e não ordenada. É adequado para aplicações que priorizam velocidade e baixa latência em detrimento da fiabilidade absoluta, enquanto o TCP é preferido em situações que exigem uma entrega fiável e ordenada dos dados.

Conclusão

Este relatório apresentou uma análise abrangente da implementação de um cliente TFTP em Python, destacando sua funcionalidade, design e mecanismos de tratamento de erros. O cliente TFTP demonstrou ser uma ferramenta eficaz para a recuperação e transferência de arquivos entre um cliente e um servidor TFTP.

Durante a análise, exploramos as principais operações do protocolo TFTP, como solicitação de leitura, solicitação de escrita, transferência de dados, confirmação e tratamento de erros. Com base nisso, desenvolvemos um cliente robusto capaz de estabelecer comunicação com um servidor TFTP e executar as operações desejadas com eficiência.

Uma das principais características do cliente TFTP é sua capacidade de lidar com erros e exceções de maneira adequada. Implementamos classes personalizadas de exceção para tratar de situações como erros de protocolo, erros de rede e erros específicos do TFTP. Isso garante uma experiência confiável e ajuda a identificar e solucionar problemas de forma rápida e eficaz.

O cliente TFTP também oferece uma interface de linha de comando (CLI) intuitiva, permitindo que os usuários interajam facilmente com o cliente e forneçam os parâmetros necessários para as operações de recuperação e transferência de arquivos. A CLI é flexível e suporta a especificação de diferentes portas de servidor, arquivos de origem e destinos, facilitando a personalização das operações conforme as necessidades do usuário.

Durante o processo de implementação, foram realizadas modificações para aprimorar a funcionalidade do cliente. Essas modificações incluíram a implementação de tratamento de tempo limite para evitar bloqueios e melhorias no tratamento de erros durante a comunicação em rede. Essas mudanças garantiram maior robustez e confiabilidade do cliente TFTP.

Em resumo, a implementação do cliente TFTP em Python mostrou-se bem-sucedida e capaz de cumprir seu propósito de transferência de arquivos eficiente entre um cliente e um servidor TFTP. O cliente oferece recursos avançados, como tratamento de erros, CLI amigável e adaptabilidade a diferentes configurações de rede.

No entanto, sempre há espaço para melhorias contínuas. Para futuras iterações, seria interessante explorar a implementação de recursos adicionais, como suporte a transferência em modo binário, criptografia de dados e suporte a outros protocolos de transferência de arquivos. Essas melhorias tornariam o cliente TFTP ainda mais poderoso e versátil em diferentes cenários de uso.

Em conclusão, o cliente TFTP em Python oferece uma solução confiável e eficiente para transferência de arquivos em ambientes de rede. Sua implementação cuidadosa e funcionalidade abrangente fazem dele uma ferramenta valiosa para profissionais que necessitam de transferência de arquivos simplificada e segura.

Bibliografia

https://www.reddit.com/

https://www.wikipedia.org/

https://python.hotexamples.com

https://wiki.python.org/moin/tftp

https://www.youtube.com/watch?v=27qfn3Gco00

https://www.youtube.com/watch?v=aTS7FBz3KLI