Parallel Downloader

Parallel Downloader is a Rust-based command-line tool aimed at creating a highly efficient download manager. The project utilizes parallel TCP connections to split files into parts, download them concurrently, and merge them efficiently. Our objective is to leverage Rust's concurrency and memory safety to achieve enhanced speed and reliability in file downloads. It supports .jpg, .png, .gif files.

Modules Overview

The project is organized into several modules, each responsible for a specific aspect of the download process:

Configuration (config.rs)
Connection Management (connection.rs)
DNS Resolution (dns.rs)
Download Management (downloader.rs)
Error Handling (error.rs)
HTTP Request Handling (http.rs)
Main Execution Flow (main.rs)

1. Configuration (config.rs)

This module defines the DownloadConfig struct, which holds all the necessary settings to initiate a file download. The settings include the URL of the file, the desired output filename, and the number of concurrent connections to use for downloading.

Key Components:
DownloadConfig struct: Stores download parameters.
new function: Creates and initializes a new DownloadConfig instance.
Example Usage: To configure a download, the DownloadConfig is initialized with the file's URL, the output path for the downloaded file, and the number of parts to divide the file into for downloading.

2. Connection Management (connection.rs)

Responsible for establishing secure connections over the internet using TLS (Transport Layer Security). This ensures that the data transferred during the download process is encrypted and secure.

Key Functions:
establish_tls_connection: Sets up a TLS-secured TCP connection to the specified hostname and IP address.
Example Usage: The function is used to create a secure channel before starting the file download, ensuring that all data transferred is encrypted.

3. DNS Resolution (dns.rs)

Handles the translation of hostnames (like www.example.com) into their respective IP addresses. This step is crucial for the connection setup as IP addresses are required to establish network connections.

Key Function:
get_request_ip: Resolves a hostname to its corresponding IP address.
Example Usage: Before establishing a connection, the hostname of the file server is resolved to an IP address using this function.

4. Download Management (downloader.rs)

Orchestrates the entire file download process. It splits the file into multiple parts, manages concurrent downloads of these parts, and finally merges them into a single file.

Key Components:
DownloadManager struct: Manages the download process.
download: Initiates the downloading of file parts.
merge_parts: Combines downloaded parts into a single file.
Example Usage: The DownloadManager uses the configuration settings to manage the download, ensuring parts are downloaded simultaneously and combined correctly.

5. Error Handling (error.rs)

Defines a custom error type, DownloaderError, which encapsulates various kinds of errors that may occur during the download process, such as IO errors, TLS handshake failures, and DNS resolution issues.

Key Component:
DownloaderError enum: Categorizes and describes potential errors.

6. HTTP Request Handling (http.rs)

Provides functionality for sending HTTP HEAD requests to retrieve metadata about the file, such as its size and whether the server supports range requests (partial content delivery).

Key Function:
send_head_request: Sends an HTTP HEAD request to gather file metadata.
Example Usage: Used to determine the size of the file and whether it can be downloaded in parts.

7. Main Execution Flow (main.rs)

Contains the main function, which is the entry point of the application. It handles user inputs, initiates the download process based on user preferences, and manages application flow.

Key Functions:
handle_download: Guides the user through setting up a download.
main: Manages the application's operation cycle.
Example Usage: Upon execution, the program prompts the user to enter the download URL and other preferences, then proceeds to download the file as configured.

Crates Dependencies

The project uses the following crates to help achieve the desired functionality:

- Standard Library (std): Rust's standard library is used for foundational functionalities like networking (std::net), file handling (std::fs), multithreading (std::thread), and input/output operations (std::io).

- native-tls: This crate provides bindings to native TLS libraries, allowing the downloader to establish secure connections using the underlying system's TLS implementation. It simplifies handling encrypted communication for file downloads over HTTPS.

- url: The url crate is utilized for URL parsing and manipulation. It provides robust tools to handle, construct, and normalize URLs, ensuring that the URLs provided for downloading are correctly formatted and processed.

Usage Instructions

To compile and run the project:

- cargo build
- cargo run
- input url to download
- input number of connections
- input output filename

To run the project using docker

- docker build -t parallel-downloader .
- docker run -it --init -v $(pwd)/downloads:/downloads parallel-downloader
- input url to download
- input number of connections
- input output filename

Example urls to test the application:

1. https://cobweb.cs.uga.edu/~perdisci/CSCI6760-F21/Project2-TestFiles/Uga-VII.jpg
2. https://cobweb.cs.uga.edu/~perdisci/CSCI6760-F21/Project2-TestFiles/story_hairydawg_UgaVII.jpg
3. https://www.nasa.gov/wp-content/uploads/2024/10/ksc-20240819-ph-jbs01-0022orig.jpg
4. https://www.nasa.gov/wp-content/uploads/2024/11/iss070e028324orig.jpg
5. https://filesampleshub.com/download/image/png/sample3.png
6. https://cobweb.cs.uga.edu/~perdisci/CSCI6760-F21/Project2-TestFiles/topnav-sport2_r1_c1.gif

Note: The application works perfectly with urls which have file extension at the end