Projekt PSI 2023Z

Skład zespołu nr.21

- Damian Pałyska
- Michał Bogiel
- Mariusz Pakulski
- Jan Kowalczewski

Temat zadania: Serwer plików

Lider zespołu – Mariusz Pakulski (01169219@pw.edu.pl)

Treść zadania

Serwer plików, udostępniający funkcjonalność pobierania plików przez klientów, łączących się za pomocą protokołu TCP. Serwer potrafi obsługiwać wielu klientów jednocześnie, dzięki zastosowaniu mechanizmu wielowątkowości.

Interpretacja treści zadania (doprecyzowanie)

Projekt ma na celu stworzenie serwera plików, który umożliwia użytkownikom pobieranie plików przez połączenia TCP. Kluczowe aspekty projektu to:

1. Protokół TCP: Serwer wykorzystuje protokół TCP, co gwarantuje niezawodność przesyłania danych. TCP zapewnia integralność danych, potwierdzenia odbioru oraz kontrolę przepływu, co jest kluczowe dla bezpiecznego przesyłania plików.
2. Wielowątkowość: Serwer wykorzystuje mechanizm wielowątkowości, aby umożliwić równoczesną obsługę żądań od wielu klientów. Dzięki temu, serwer może efektywnie zarządzać równoległymi połączeniami, zwiększając swoją skalowalność i wydajność.
3. Pobieranie Plików: Podstawową funkcją serwera jest umożliwienie klientom pobierania plików. Klienci będą wysyłać żądania określające, które pliki chcą pobrać, a serwer będzie odpowiedzialny za ich przesyłanie.

Projekt będzie składał się z 2 programów: serwera oraz klienta. Serwer będzie udostępniał pliki znajdujące się w jego folderze, zaś klient będzie zapisywał pobrane pliki w swoim folderze.

Krótki opis funkcjonalny – „black-box”

- Serwer plików – działa jako centralny węzeł w sieci, realizujący następujące funkcje:
    - Przechowywanie plików:
        - Przechowuje pliki dostępne do pobrania
        - Zarządza dostępem do nich
    - Obsługa połączeń TCP:
        - Nasłuchuje na określonym porcie na połączenia TCP przychodzące od klientów
        - Ustanawia z nimi połączenia
    - Wielowątkowa obsługa żądań:

        - Dla każdego połączenia klienta, tworzy osobny wątek, umożliwiając równoczesną obsługę wielu żądań
        - Każdy wątek jest odpowiedzialny za obsługę interakcji z pojedynczym klientem
    - Proces przesyłania plików:
        - Po otrzymaniu żądania od klienta, inicjuje i kontroluje przesyłanie pliku do klienta
    - Wyświetlanie listy plików:
        - Po otrzymaniu żądania "ls" od klienta wyświetla zawartość wskazanego w argumencie katalogu. Domyślnie jest to główny katalog serwera
        - Po otrzymaniu żądania "tree" od klienta wyświetla drzewo folderów dla wskazanego w argumencie katalogu. Domyślnie dla głównego katalogu serwera

- Klient – aplikacja kliencka, która komunikuje się z serwerem, realizując następujące funkcje:

    - Inicjowanie połączenia
        - Nawiązuje połączenie TCP z serwerem na określonym porcie.
        - Wysyła żądanie pobrania określonego pliku.
    - Odbiór plików
        - Po otrzymaniu pliku od serwera, zapisuje go w lokalnym systemie plików.
    - Interfejs użytkownika
        - Umożliwia użytkownikowi wprowadzenie nazwy pliku do pobrania
        - Umożliwia użytkownikowi wyświetlenie zawartości lub drzewa folderów dla wskazanego katalogu

Opis i analiza poprawności stosowanych protokołów komunikacyjnych

Dla naszego projektu kluczowe znaczenie ma zastosowanie protokołu TCP wraz z IP. TCP zapewnia niezawodność i kontrolę przepływu danych, co jest niezbędne dla przesyłania plików, podczas gdy IP umożliwia właściwe adresowanie i routing w sieci. Razem tworzą one solidną podstawę dla bezpiecznego i efektywnego przesyłania danych w systemie serwera plików.

Protokół TCP (Transmission Control Protocol)

- Niezawodność: TCP jest protokołem zorientowanym na połączenie, który zapewnia niezawodne przesyłanie danych poprzez potwierdzenia i retransmisję utraconych pakietów.
- Kontrola Przepływu: TCP zarządza kontrolą przepływu danych, co zapobiega przeciążeniu sieci i zapewnia równomierne przesyłanie danych.
- Zastosowanie w Projekcie: W projekcie serwera plików, TCP jest wykorzystywany do nawiązywania stabilnych połączeń z klientami, co jest kluczowe dla przesyłania plików. TCP gwarantuje, że pliki są przesyłane kompletnie i w odpowiedniej kolejności.

Protokół IP (Internet Protocol)

- Adresacja i Routing: IP odpowiada za adresowanie i dostarczanie pakietów danych do odpowiedniego miejsca docelowego w sieci.
- Bezstanowość i Niezależność: Każdy pakiet danych traktowany jest niezależnie, co pozwala na elastyczność w routingu przez różne ścieżki w sieci.
- Zastosowanie w Projekcie: Protokół IP jest wykorzystywany w połączeniu z TCP do adresowania i dostarczania danych między serwerem a klientami. Odpowiedź serwera

Zapytanie klient-serwer

Nazwa	Typ danych
polecenie	str
ścieżka do pliku	str

JSON: { "command": "get/ls/tree", "path": "ścieżka" }

Odpowiedź serwer-klient

W przypadku pobierania plików:

W przypadku poprawnego zapytania klienta:

Nazwa	Typ danych
status	str
wielkość pliku (B)	str

JSON: { "status": "ok", "size": rozmiar }

A następnie przesłanie samego pliku:

Nazwa	Typ danych
plik	ciąg bajtów

W przypadku niepoprawnego zapytania klienta:

Nazwa	Typ danych
status	str
komunikat	str

JSON: { "status": "error", "message": "opis błędu" }

możliwe błędy:

- "Niepoprawna ścieżka"
- "Plik nie znaleziony"

W przypadku wyświetlania listy plików

W przypadku poprawnego zapytania klienta:

Nazwa	Typ danych
status	str
lista plików	str

JSON: { "status": "ok", "data": "dane" }

W przypadku niepoprawnego zapytania klienta:

Nazwa	Typ danych
status	str
komunikat	str

JSON: { "status": "error", "message": "opis błędu" }

możliwe błędy:

- "Niepoprawna ścieżka"
- "Katalog nie znaleziony"

W przypadku wyświetlania drzewa plików

W przypadku poprawnego zapytania klienta:

Nazwa	Typ danych
status	str
drzewo plików	str

JSON: { "status": "ok", "data": "dane" }

W przypadku niepoprawnego zapytania klienta:

Nazwa	Typ danych
status	str
komunikat	str

JSON: { "status": "error", "message": "opis błędu" }

możliwe błędy:

- "Niepoprawna ścieżka"
- "Katalog nie znaleziony"

Planowany podział na moduły i struktura komunikacji

Moduły

- Moduł Serwera:
    - start_server(host, port): nasłuchuje na połączenia TCP na określonym porcie i tworzy nowe wątki
    - handle_connection(connection,address): wywoływana w każdym nowym wątku, odbiera komunikaty od klienta, wywołuje funkcje sprawdzające czy zapytanie jest poprawne, po czym wysyła dane do klienta, i czeka na kolejne zapytanie / zakończenie połączenia.

    - verify_file_path(path) - sprawdza czy dane do klienta są legalne (czy ścieżka do pliku nie wychodzi poza katalog)
    - send_file(connection, path) - przesyła plik do klienta
    - send_ls(connection, path): przesyła do klienta zawartość wskazanego katalogu (json serializowany do stringa, kodowany jako ciąg bajtów)
    - send_tree(connection, path): przesyła do klienta drzewo folderów dla wskazanego katalogu (j.w.)

- Moduł Klienta:

    - main(): łączy się z serwerem i w pętli prosi użytkownika o podanie ścieżki pliku do pobrania
    - send_request(host, port, command, path): wysyła polecenie (get, ls lub tree) do serwera wraz ze ścieżką (do pliku lub katalogu)
    - receive_json_response(socket): dekoduje otrzymany ciąg bajtów do stringa i odzyskuje z niego przesłanego jsona z metadanymi (status i ew. roziar) i/lub danymi (dla ls i tree)
    - receive_file_data(socket, path, file_size): pobiera od serwera plik i zapisuje go pod wskazaną ściezkę

Struktura Komunikacji

Komunikacja między klientem a serwerem odbywa się przez połączenia TCP, gdzie serwer obsługuje każde połączenie w osobnym wątku. Klient inicjuje połączenie, i wysyła żądania dotyczące plików, po czym odbiera i zapisuje pliki przesłane przez serwer lub wyświetla otrzymaną listę plików

Zarys koncepcji implementacji

- Język Programowania: Python3 (oferuje wsparcie dla sieciowych operacji i wielowątkowości).
- Biblioteki:
    - socket (dla obsługi TCP)
    - threading (dla wielowątkowości)

Najważniejsze rozwiązania funkcjonalne

1. Protokoł TCP/IP: ponieważ gwarantuje niezawodność przesyłania danych, integralność danych, potwierdzenie odbioru oraz kontrolę przepływu. Pozwala to nawiązywać stabilne i trwałe połączenia między klientem a serwerem, co jest kluczowe dla bezpiecznego przesyłania plików. IP natomiast pozwala na adresowanie i routing pakietów w sieci.
2. Wielowątkowość: Serwer wykorzystuje mechanizm wielowątkowości, aby umożliwić równoczesną obsługę żądań od wielu klientów. Dzięki temu, serwer może efektywnie zarządzać równoległymi połączeniami, zwiększając swoją skalowalność i wydajność.
3. Bezpieczeństwo dostępu do plików i katalogów: Weryfikcja ścieżki zapobiega dostępowi do plików lub folderów poza określonym katalogiem.
4. Binarny dostęp do plików - serwer odczytuje pliki w trybie binarnym, a klient w trybie binarnym je zapisuje. Tryb binarny pozwala wiernie przesłać całą zawartość pliku.
5. Przesyłanie metadanych i danych (oprócz samej zawartości pliku) w formacie JSON - ponieważ wygodnie można zapisywać do niego informacje i odczytywać je z niego oraz posiada on wbudowane metody do serializacji (z JSONa do stringa) i deserializacji (ze stringa do JSONa)

Struktury danych

Format Komunikatów JSON

- Żądania od klienta: Struktura JSON zawierająca klucze command i path, które określają rodzaj żądania (np. get, ls, tree) i ścieżkę do pliku lub katalogu.
- Odpowiedzi od serwera: JSON z kluczami status, message (w przypadku błędów), data (czyli dane; dla odpowiedzi ls i tree) oraz size (dla odpowiedzi get).

Kluczowe funkcje

Serwer:

- start_server(host, port): Inicjuje serwer na określonym hoście i porcie. Ustawia gniazdo sieciowe, wiąże je z adresem i zaczyna nasłuchiwać przychodzących połączeń. Przy każdym nowym połączeniu uruchamia nowy wątek za pomocą funkcji handle_client. Jest punktem wejścia dla serwera, zarządzającym rozpoczęciem procesu obsługi klientów.
- handle_client(connection, address): Główna funkcja wątku serwera, zarządzająca połączeniem z pojedynczym klientem. Odbiera dane JSON, interpretuje je i wywołuje odpowiednie funkcje w zależności od rodzaju żądania.
- send_file(connection, requested_path): Przesyła plik do klienta. Najpierw weryfikuje ścieżkę do pliku, a następnie wysyła metadane (rozmiar pliku) i zawartość pliku w blokach o rozmiarze 4096 bajtów.
- send_ls(connection, requested_path) i send_tree(connection, requested_path): Odpowiadają za generowanie i wysyłanie listy plików (ls) oraz struktury drzewa katalogów (tree). Funkcje te używają os.listdir do pobierania informacji o plikach i katalogach, a następnie formatują te dane do przesłania jako odpowiedź JSON.
- verify_file_path(requested_path): Funkcja serwera zapewniająca, że żądanie dostępu do pliku nie wychodzi poza zdefiniowany katalog (np. katalog główny serwera). Jest to kluczowe dla zapewnienia bezpieczeństwa, chroniąc przed nieautoryzowanym dostępem do systemu plików serwera.

Klient:

- main(): Główna funkcja wykonawcza klienta. Prosi użytkownika o wprowadzenie adresu IP serwera, po czym w pętli umożliwia użytkownikowi wybór komendy i wprowadzenie odpowiednich danych (np. ścieżki do pliku). Zależnie od wybranej komendy, wywołuje funkcję send_request z odpowiednimi argumentami. Stanowi interfejs użytkownika dla aplikacji klienta, umożliwiając interakcję z serwerem.
- send_request(server_host, server_port, command, path): Główna funkcja klienta do wysyłania żądań do serwera. Tworzy połączenie TCP, wysyła żądanie w formacie JSON i odbiera odpowiedź.
- receive_json_response(client_socket): Odbiera odpowiedź JSON od serwera. Czyta dane z gniazda sieciowego i konwertuje je na format JSON. Funkcja ta jest używana do odbioru metadanych odpowiedzi przed faktycznymi danymi (np. przed odbiorem pliku). Kluczowa dla interpretacji odpowiedzi serwera, szczególnie przy rozróżnianiu między różnymi rodzajami odpowiedzi (sukces/błąd, komunikat/rozmiar/dane).
- receive_file_data(client_socket, save_path, file_size): Odbiera dane pliku od serwera i zapisuje je lokalnie. Funkcja ta czyta dane z gniazda sieciowego w blokach i zapisuje je w pliku aż do osiągnięcia zgłoszonego rozmiaru pliku.

Kluczowe biblioteki

- socket: Obydwa moduły, serwer i klient, używają gniazd sieciowych (socket) do tworzenia połączeń TCP/IP. Gniazda te pozwalają na wysyłanie i odbieranie danych przez sieć. Kluczowe dla całej

komunikacji sieciowej projektu, od nawiązywania połączeń po przesyłanie danych.
- json: Oba moduły używają JSONa do formatowania i interpretacji danych komunikatów. Zapewnia to spójny i łatwo parsowalny format wymiany danych między serwerem a klientem. Umożliwia to strukturalne i czytelne przesyłanie poleceń oraz odpowiedzi, co jest kluczowe dla logicznego przepływu danych.
- os: zapewnia interakcję z systemem operacyjnym. Umożliwia wykonywanie wielu operacji związanych z systemem plików, takich jak listowanie katalogów, przetwarzanie ścieżek, sprawdzanie istnienia plików, itp. Używana jest na serwerze m.in. do listowania zawartości katalogów (os.listdir) w realizacji komend ls i tree. Pomaga także w weryfikacji ścieżek do plików (np. przy użyciu os.path.abspath i os.path.join), co jest kluczowe dla bezpieczeństwa serwera.
- threading: umożliwia tworzenie i zarządzanie wątkami. Wątki są lżejszymi jednostkami wykonawczymi niż procesy i umożliwiają równoległe wykonywanie zadań w obrębie jednego procesu. Używany do obsługi wielowątkowości. Serwer tworzy nowy wątek dla każdego połączenia klienta, co pozwala na równoczesne obsługiwania wielu żądań. To zapewnia lepszą skalowalność i wydajność serwera.

Interfejs użytkownika

Interfejs użytkownika (po stronie klienta) opiera się na interfejsie wiersza poleceń (CLI - Command Line Interface), który zapewnia prostą i efektywną komunikację między użytkownikiem a systemem.

1. Po uruchomieniu, klient prosi o podanie adresu IP serwera. Można użyć domyślnego adresu (localhost) lub wprowadzić inny adres. Przykład: Wprowadź adres IP serwera [127.0.0.1]:

2. Wprowadzanie komend: Po nawiązaniu połączenia, użytkownik ma dostęp do następujących komend: get, ls, tree, exit. Przykład: Dostępne komendy: get, ls, tree, exit

3. Pobieranie plików (get):

Użytkownik może poprosić o pobranie pliku, wpisując get, a następnie ścieżkę do żądanego pliku. Przykład: Wprowadź ścieżkę do pliku: [ścieżka/pliku] Po otrzymaniu odpowiedzi od serwera, jeśli plik jest dostępny, użytkownik wprowadza nazwę pliku, pod którą chce go zapisać lokalnie. Przykład: Podaj nazwę pliku do zapisu (opcjonalnie ze ścieżką): [ścieżka/pliku]

4. Wyświetlanie zawartości katalogu (ls):

Umożliwia wyświetlenie listy plików i katalogów w określonym katalogu serwera. Przykład: Wprowadź ścieżkę: [ścieżka/katalogu]

5. Wyświetlanie drzewa Katalogów (tree): Wyświetla strukturę katalogów serwera w formie drzewa, zaczynając od wskazanego katalogu. Przykład: Wprowadź ścieżkę: [ścieżka/katalogu]

6. Zakończenie sesji (exit): Użytkownik może zakończyć sesję i zamknąć klienta, wpisując exit.

Po każdym żądaniu, klient odbiera odpowiedź od serwera i wyświetla ją użytkownikowi. W przypadku błędów (np. plik nie istnieje, błąd ścieżki), wyświetlany jest odpowiedni komunikat błędu.

Opis wykorzystanych narzędzi

Język programowania: Python 3. Wybrany ze względu na prostotę składni, wsparcie dla sieciowych operacji I/O, wielowątkowości oraz obszerne biblioteki standardowe, które ułatwiają pracę z protokołami sieciowymi,

przetwarzaniem danych i interakcją z systemem operacyjnym.

Biblioteki Programistyczne:

- socket: Biblioteka wykorzystywana do obsługi połączeń sieciowych w Pythonie. Umożliwia tworzenie gniazd sieciowych, które są fundamentem komunikacji TCP/IP w projekcie.
- threading: Używana do implementacji wielowątkowości. Dzięki niej serwer może obsługiwać wielu klientów równocześnie, każdego w osobnym wątku.
- os: Zapewnia funkcje interakcji z systemem operacyjnym, w tym zarządzanie plikami i katalogami. Jest kluczowa dla operacji takich jak listowanie zawartości katalogów oraz sprawdzanie istnienia i dostępu do plików.
- json: Umożliwia serializację i deserializację danych w formacie JSON, który jest wykorzystywany do komunikacji między klientem a serwerem.

Docker: Platforma do tworzenia, uruchamiania i zarządzania kontenerami aplikacji. Kontenery Docker umożliwiają izolację aplikacji, co sprawia, że są one bardziej przenośne, spójne i łatwe w konfiguracji. Został użyty do wykonania testowania w izolowanym środowisku: Docker pozwala na uruchomienie serwera i klienta w oddzielnych, izolowanych środowiskach, co jest szczególnie przydatne do testowania interakcji między nimi. Dzięki temu możliwe jest symulowanie różnych warunków sieciowych i środowiskowych bez wpływu na główny system operacyjny. Docker zapewnia, że aplikacja działa tak samo w każdym środowisku, ponieważ wszystkie zależności są zdefiniowane i zawarte w kontenerze. To ułatwia testowanie i eliminuje problem "u mnie działa".

Pliki konfiguracyjne (Docker)

[TODO]

Opis testów i wyników testowania (z logami)

[TODO] Przeprowadzzono testy... Oto logi z testów... Wyniki testowania wskazują na... Bla bla bla...

Jeśli testy wykryły błędy to: znalezienie błędów w programie pokazuje, że udało się przygotować skuteczne testy... Jeśli nie to: mimo rewelacyjnych, wspaniałych, gruntownych testów, nie wykryto błędów, czyli program jest super...