Dokumentacja końcowa projektu: Zdalny serwer plików z obsługą SCTP/TCP, IPv4/IPv6 oraz dodatkowymi funkcjonalnościami.

Wykonawcy:

Mateusz Oleksy (416651)

Paweł Urban (415846)

Prowadzący:

mgr. inż. Karol Salwik

dr. inż. Janusz Gozdecki

1. Wprowadzenie

Projekt polega na stworzeniu zdalnego serwera plików, który umożliwia klientowi wykonywanie operacji na plikach i folderach znajdujących się po stronie serwera. Komunikacja odbywa się za pomocą protokołów SCTP lub TCP, z obsługą zarówno IPv4, jak i IPv6. Serwer i klient wspierają autoryzację przy użyciu loginu i hasła oraz posiadają zaimplementowane mechanizmy obsługi sygnałów oraz timeoutów. Aplikacja i klienta jest przykładową implementacją usługi rodzaju “Dysk Google” lub usług “One Drive” od korporacji Microsoft.

---

2. Funkcjonalności projektu:

2.1. Komunikacja serwer-klient

- Protokół: SCTP lub TCP. Wybór polega na wybraniu odpowiedniego protokołu dla sctp przy inicjalizacji serwera.
- Obsługa IPv4 i IPv6, z możliwością wymuszenia użycia wyłącznie IPv6 za pomocą opcji IPV6_ONLY.
- Mapowanie nazw domenowych na adresy IP odbywa się za pomocą pliku /etc/hosts na komputerze klienta.

2.2. Timeout

Timeout dla klienta jest to 5 sekund dla funkcji wysyłającej oraz obierającej z gniazda

- Timeout dla serwera jest to 60 sekund dla funkcji cytającej z gniazda

2.3. Obsługa sygnałów

- **Klient:** Obsługuje sygnał SIGPIPE, co zapobiega nieoczekiwanym przerwaniu połączenia w przypadku zerwania połączenia przez serwer.
- **Serwer:** Obsługuje sygnały SIGPIPE oraz SIGCHLD, co pozwala na odpowiednie zarządzanie procesami potomnymi i uniknięcie tworzenia procesów zombie.

  Dodatkowe zabezpieczenia przed błędami, sprawdzanie poprawności i dostępności pliku, sprawdzanie zwracanego kodu funkcji send/recv. Postanowiliśmy jednak na zabezpieczenia w postaci ustawienia timeout-ów na gnieździe dla funkcji blokowalnych.

2.4. Logowanie klienta

- Logowanie odbywa się za pomocą loginu i hasła przesyłanych do serwera w standardowej funkcji send(); Możliwe przyszłe ulepszenie polegające na dodaniu zabezpieczenia w postaci TLS.
- Serwer weryfikuje dane logowania przed udzieleniem dostępu do zasobów. Każdy folder to unikalny nickname użytkownika. Nickname nie może się powtarzać.

---

3. Obsługiwane komendy

Serwer obsługuje szereg komend przesyłanych przez klienta, które pozwalają na zarządzanie plikami i folderami.

3.1. Komenda LS

- Wyświetla listę plików i folderów znajdujących się w bieżącym katalogu klienta. W tym którym uruchomiliśmy program.

3.2. Komenda SHOW [nazwa_pliku]

- Wyświetla zawartość wskazanego pliku po stronie klienta, zawartego w danym folderze.

3.3. Komenda CAT [nazwa_pliku]

- Wyświetla zawartość wskazanego pliku po stronie serwera w folderze klienta.

3.4. Komenda UPLOAD [nazwa_folderu]/[pliku]

- Przed wysłaniem folder jest pakowany w archiwum .zip.
- Wysłany jest plik lub folder wskazany komendą do katalogu klienta.

3.5. Komenda DOWNLOAD [pliku]

- Dany plik jest pobierany z serwera do folder klienta, czyli folder, w którym znajduje się plik wykonywalny aplikacji klienta

3.6. Komenda LIST

- Wypisuje zawartość folderu klienta po stronie serwera.

3.7. Komenda DELETE [plik]

- Usuwa dany plik po stronie serwera w katalogu klienta

3.8. Komenda QUIT

- Kończy połączenie klienta z serwerem.

---

4. Nawiązanie połączenia

- Można zalogować się podając odpowiedni protokół SCTP/TCP wybrany przy incjalizacji serwera jako pierwszy argument po nazwie programu klienta.
- Wymagany jest argument trzeci czyli adres IP serwera. Jeżeli chcemy zrobić to poprzez odzwierciedlenie w nazwanie DNS z pliku /etc/hosts z predefiniowaną nazwą serwera to musimy podać odpowiedni protokół jako pierwszy argument, jako drugi 0, a następnie nazwę serwera.
- Po połączeniu z serwerem są do wyboru trzy komendy LOGIN, REGISTER, QUIT
- Po wybraniu LOGIN, bądź REGISTER podajemy odpowiedni login i hasło. Po udanej rejestracji jesteśmy automatycznie zalogowani.
- Nie ma możliwości zalogowania się dwóch użytkowników o tych samych nazwach.

---

5. Implementacja

5.1. Konfiguracja usługi:

Projekt został zaimplementowany w języku C z wykorzystaniem gniazd sieciowych (sockets) oraz bibliotek systemowych. Kod serwera został dostosowany do ciągłego działania w powłoce linuxa jako usługa. Poniżej przedstawiamy konfigurację usługi w systemie LINUX.

W katalogu linux etc/systemd/system utworzyliśmy plik server_ftp.service o treści:

[Unit]

Description=Server do wymiany plików z klientem

After=network.target

[Service]

ExecStart= /home/matt2/Downloads/./server.c sctp

Restart=always

User=root

Group=root

[Install]

WantedBy=multi-user.target

W bloku [init], definiujemy odpowiedni opis usługi, After=network.target powoduje uruchomienie skryptu po inicjalizacji sieci.

W bloku [service] podajemy odpowiednią ścieżkę do katalogu, w którym znajduje się plik wykonywalny serwera. Restart=always powoduje ponowne uruchomienie skryptu w przypadku zwrócenia błędu przez program. Definiujemy też z jakiego poziomu, grupy, użytkownika mamy zainicjować usługę.

WantedBy=multi-user.target w bloku [Install] powoduje uruchamianie skryptu po włączeniu systemu LINUX.

5.2. Konfiguracja synchronizacji klientów:

Aktualnie zapisanego użytkownika serwer zapisuje do pliku conf/login. Wszystkie dane użytkownika zapisane są w conf/configuration. Jest jeszcze pomocniczy plik temp w folderze conf o nazwie temp, który służy za plik edytowalny serwerowi, np. do przepisania aktywnych użytkowników. Przy inicjalizacji serwera plik login jest czyszczony tak aby nie doszło do sytuacji gdy po np. restarcie serwer nie pozwoli zalogować się jakiemuś użytkownikowi. Na początku kodu serwera można podać ścieżkę (zmienna source) gdzie znajduje się folder conf, a nim pliki configuration i login w formacie /home/matt2/Downloads/.

5.3. Kompilacja:

Kompliacja przebiega domyślnie za pomocą komendy gcc server.c -o server np. z tym wyjątkiem, że przy uruchamianiu protokołu SCTP należy pamiętać o bibliotece #include <netinet/sctp.h>, w naszym przypadku użyliśmy komendy sudo modprobe sctp oraz zainstalowaliśmy bibliotekę libsctp-dev na naszym maszynach wirtualnych służących do testowania.

---

6. Podsumowanie

Projekt zdalnego serwera plików zrealizowano zgodnie z założeniami. Kluczowe funkcjonalności to obsługa różnych protokołów i wersji IP, timeout oraz dodatkowe komendy operujące na plikach. Obsługa sygnałów oraz autoryzacja użytkowników zapewniają stabilność i bezpieczeństwo działania systemu.

Możliwe dalsze usprawnienia:

- Dodanie szyfrowania przesyłanych danych (np. TLS).
- Wprowadzenie bardziej zaawansowanego systemu logowania zdarzeń po stronie serwera.
- Rozszerzenie zestawu dostępnych komend.