Techniki internetowe

Dokumentacja wstępna projektu "TinDox"

Jakub Mazurkiewicz	Damian Piotrowski	Anna Pyrka	Łukasz Reszka

Semestr 21Z

Spis treści

1 Cel projektu **3**

2 Protokół TDP **3**
- 2.1 Opis 3
- 2.2 Słownik pojęć 3
- 2.3 Metoda autoryzacji 4
- 2.4 Poruszanie się po katalogach 4
- 2.5 Wyświetlanie aktualnego katalogu 4
- 2.6 Wypisywanie katalogów 4
- 2.7 Tworzenie katalogów 4
- 2.8 Tworzenie plików 5
- 2.9 Usuwanie plików i katalogów 5
- 2.10 Zmienianie nazwy plików i katalogów 6
- 2.11 Kopiowanie lub przenoszenie plików między katalogami 6
- 2.12 Pobieranie plików 7
- 2.13 Ładowanie plików 8
- 2.14 Struktura komunikatów 10
  - 2.14.1 Polecenia dla serwera 10
  - 2.14.2 Odpowiedzi serwera na polecenia 11
  - 2.14.3 Transmisja danych 11

3 Serwer **12**
- 3.1 Słownik pojęć 12
- 3.2 Wymagania funkcjonalne 12
- 3.3 Wymagania niefunkcjonalne 12
- 3.4 Polecenia linii komend 13
- 3.5 Narzędzia i biblioteki 13

4 Klient mobilny **13**
- 4.1 Opis klienta 13
- 4.2 Narzędzia i biblioteki 13

5 Klient okienkowy **14**
- 5.1 Opis klienta 14

5.2 Narzędzia i biblioteki 14

6 Klient konsolowy 14
6.1 Opis klienta 14
6.2 Narzędzia i biblioteki 14

7 Metody testowania 14

8 Zespół 15
8.1 Środowisko deweloperskie 15
8.2 Podział pracy 15

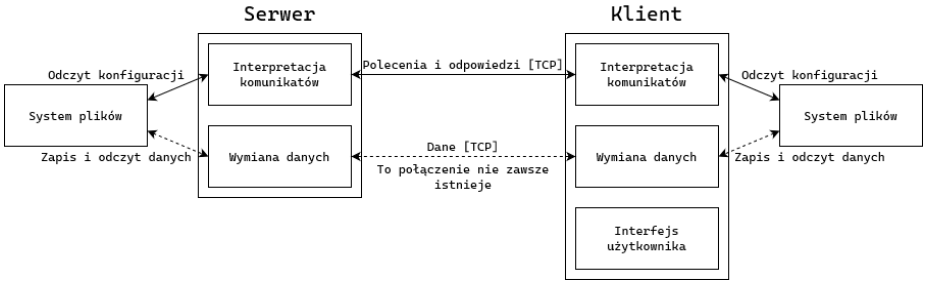
1. Cel projektu

Celem projektu jest opracowanie protokołu do wymiany plików przez sieć IPv4. Zakłada się także stworzenie wydajnego serwera oraz klientów na wybrane platformy.

2. Protokół TDP

2.1. Opis

TDP to protokół komunikacyjny typu serwer–klient wykorzystujący protokół sterowania transmisją (TCP). Umożliwia dwukierunkowy transfer plików oraz przeglądanie katalogów znajdujących się na zdalnym dysku.



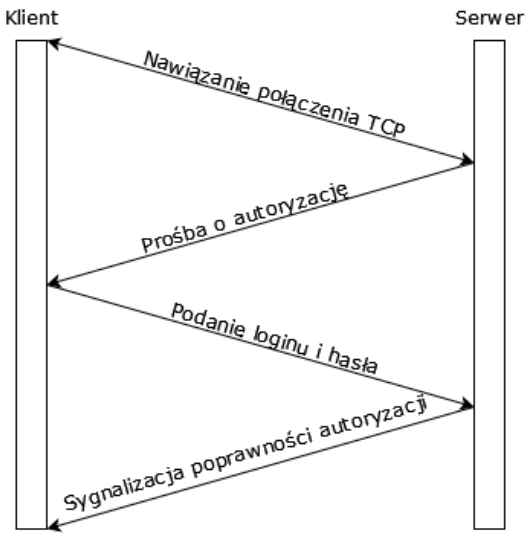
Rysunek 1: Schemat sesji inspirowany protokołem FTP.

Komunikaty w protokole wymieniane są za pośrednictwem połączenia głównego (na rysunku oznaczonym jako “Polecenia i odpowiedzi”). Dane są przesyłane za pośrednictwem dodatkowego połączenia TCP (na rysunku oznaczonym jako “Dane”) tworzonego tylko na potrzeby pobierania i ładowania plików. Po zakończeniu jednej z tych operacji połączenie nie jest od razu zamykane – może zostać użyte do kolejnych transferów.

2.2. Słownik pojęć

1. TDP – TinDox Protocol.
2. Aktualny katalog – każdy zalogowany użytkownik posiada przypisany do siebie katalog bieżący, czyli ten, w którym się aktualnie znajduje.
3. Sól – krótki, losowo wygenerowany ciąg składający się z wielkich i małych liter alfabetu łacińskiego oraz cyfr, identyfikujący pewną operację lub plik.
4. Kod operacji – losowo wygenerowany 32-bitowy kod identyfikujący pewną złożoną operację (np. pobieranie pliku).

2.3. Metoda autoryzacji



Rysunek 2: Schemat autoryzacji

W przypadku niepomyślnej autoryzacji serwer wysyła do klienta ponowną prośbę o podanie danych. Po trzech nieudanych próbach serwer zamyka połączenie.

2.4. Poruszanie się po katalogach

Klient ma możliwość przejścia do katalogu domowego lub o jeden poziom względem bieżącego katalogu. Analogicznym działaniem w systemie Linux jest instrukcja `cd`.

2.5. Wyświetlanie aktualnego katalogu

Klient ma możliwość wyświetlenia ścieżki do aktualnego katalogu. Analogicznym działaniem w systemie Linux jest instrukcja `pwd`.

2.6. Wypisywanie katalogów

Istnieją dwie możliwości:

1. Można wypisać wszystkie katalogi zaczynając od katalogu domowego (analogiczne polecenie systemowe to `tree`),
2. Można wypisać podkatalogi i pliku znajdujące się w bieżącym katalogu (analogiczne polecenie systemowe `ls`).

2.7. Tworzenie katalogów

Możliwe jest tworzenie katalogów tylko wewnątrz bieżącego katalogu, pojedynczo. Jeśli katalog o podanej nazwie już istnieje użytkownik otrzyma błąd.

Od momentu podania nazwy nowego katalogu jest ona już "zajęta", w przypadku gdy dwie osoby spróbują stworzyć w tym samym czasie katalog o tej samej nazwie, jedna z nich otrzyma błąd.

2.8. Tworzenie plików

- Scenariusz główny:
  1. Klient wysyła do serwera żądanie utworzenia pliku o wskazanej nazwie w folderze, w którym obecnie się znajduje,
  2. Serwer sprawdza, czy w danym katalogu istnieje już plik o danej nazwie,
  3. Serwer tworzy plik o podanej nazwie w odpowiednim katalogu,
  4. Serwer wysyła do klienta potwierdzenie utworzenia nowego pliku.
- Scenariusz alternatywny – próba utworzenia w "folderze klienta" pliku o tej samej nazwie co inny plik:
  1. Kroki 1 – 2 scenariusza głównego,
  2. Serwer wysyła do klienta odmowę wykonania operacji.

2.9. Usuwanie plików i katalogów

- Scenariusz główny:
  1. Klient wysyła do serwera żądanie usunięcia wskazanego pliku lub katalogu,
  2. Serwer sprawdza, czy istnieje katalog (lub plik) do usunięcia,
  3. Serwer sprawdza, czy w katalogu do usunięcia ktoś się znajduje lub czy plik do usunięcia jest aktualnie pobierany,
  4. Jeśli nie, to serwer usuwa katalog (lub plik) i wysyła potwierdzenie usunięcia do klienta.
- Scenariusz alternatywny I – w katalogu do usunięcia ktoś się znajduje lub plik do usunięcia jest pobierany:
  1. Kroki 1 – 3 scenariusza głównego,
  2. Serwer wysyła do klienta odmowę wykonania operacji.
- Scenariusz alternatywny II – próba usunięcia nieistniejącego pliku lub katalogu:
  1. Kroki 1 – 2 scenariusza głównego,
  2. Serwer wysyła do klienta odmowę wykonania operacji.

2.10. Zmienianie nazwy plików i katalogów

- Scenariusz główny:
  1. Klient wysyła do serwera żądanie zmiany nazwy pliku lub katalogu,
  2. Serwer sprawdza, czy istnieje plik lub katalog o tej samej nazwie w danym katalogu,
  3. Jeśli jest to plik, serwer sprawdza, czy jest on aktualnie pobierany,
  4. Jeśli jest to katalog, serwer sprawdza, czy aktualnie ktoś się w nim znajduje,
  5. Jeśli nie, serwer zmienia nazwę danego pliku lub katalogu i wysyła potwierdzenie do klienta.
- Scenariusz alternatywny I – istnieje już plik lub katalog o tej samej nazwie:
  1. Kroki 1 – 2 scenariusza głównego,
  2. Serwer wysyła do klienta odmowę zmiany nazwy.
- Scenariusz alternatywny II – próba zmiany nazwy pliku który jest obecnie pobierany:
  1. Kroki 1 – 3 scenariusza głównego,
  2. Serwer wysyła do klienta odmowę zmiany nazwy.
- Scenariusz alternatywny III – próba zmiany nazwy katalogu w którym ktoś się znajduje:
  1. Kroki 1 – 4 scenariusza głównego,
  2. Serwer wysyła do klienta odmowę zmiany nazwy.

2.11. Kopiowanie lub przenoszenie plików między katalogami

- Scenariusz główny:
  1. Klient wysyła do serwera żądanie skopiowania pliku o danej nazwie do zadanego katalogu,
  2. Serwer sprawdza, czy istnieje katalog, do którego chcemy skopiować plik,
  3. Serwer sprawdza, czy w danym katalogu istnieje już plik o danej nazwie,
  4. Jeśli nie, serwer kopiuje plik do podanego katalogu.
- Scenariusz alternatywny I – plik który chcemy skopiować jest w katalogu, w którym obecnie się nie znajdujemy:
  1. Kroki 1 scenariusza głównego,
  2. Serwer wysyła do klienta odmowę skopiowania pliku.
- Scenariusz alternatywny II – próba skopiowania pliku do katalogu, który nie istnieje:

1. Kroki 1 – 2 scenariusza głównego,
2. Serwer tworzy katalog o podanej nazwie, a następnie kopiuje do niego plik.

- Scenariusz alternatywny III – próba skopiowania pliku w miejsce gdzie istnieje już inny o tej samej nazwie:

  1. Kroki 1 – 3 scenariusza głównego,
  2. Serwer wysyła do klienta odmowę skopiowania pliku.

2.12. Pobieranie plików

Klient może pobrać dany plik z serwera z aktualnego katalogu pod warunkiem, że ma do tego odpowiednie uprawnienia:

- Scenariusz główny:

  1. Klient wysyła pytanie o zgodę na pobranie pliku. Zawiera się w niej przede wszystkim nazwa przesyłanego pliku,
  2. Serwer, po zweryfikowaniu uprawnień użytkownika, wysyła zgodę na pobieranie wraz z dodatkowymi informacjami (np. rozmiar pliku, jaki jest największy możliwy pakiet do odebrania, kod identyfikujący operację),
  3. Klient nawiązuje dodatkowe połączenie TCP z serwerem, które będzie wykorzystywane do przesyłania danych,
  4. Serwer dzieli plik na małe pakiety i wysyła je do klienta,
  5. Klient odbiera fragmenty od serwera i dopisuje je do pliku o nazwie "`[NAZWA-ŁADOWANEGO-PLIKU].[SÓL].partial`" reprezentującego częściowo pobrane dane.
  6. Ostatnim pakietem wysłanym przez serwer jest informacja o zakończeniu ładowania,
  7. Aplikacja klienta zmienia nazwę pliku częściowego na docelową,
  8. Utworzone wcześniej dodatkowe połączenie TCP nie jest zamykane od razu – istnieje możliwość wykorzystania go do kolejnych pobrań (lub ładowań).

- Scenariusz alternatywny I - odmowa pobierania:

  1. Krok 1 scenariusza głównego,
  2. Klient otrzymuje odmowę pobierania pliku – użytkownik dostaje informację o niepowodzeniu operacji.

- Scenariusz alternatywny II – dodatkowe połączenie TCP z serwerem zostało wcześniej zestawione:

  1. Kroki 1 – 2 scenariusza głównego,
  2. Dodatkowe połączenie już istnieje – serwer i klient będą je wykorzystywać do przesyłania danych,
  3. Kroki 4 – 8 scenariusza głównego.

- Scenariusz alternatywny III – dodatkowe połączenie TCP nie mogło zostać zestawione:
    1. Kroki 1 – 2 scenariusza głównego,
    2. Dodatkowe połączenie nie mogło zostać zestawione – użytkownik dostaje informację o niepowodzeniu operacji.
- **Scenariusz alternatywny IV – przerwanie połączenia z siecią w trakcie pobierania pliku**:
    1. Kroki 1 – 4 scenariusza głównego,
    2. Przerwanie połączenia ze strony klienta lub serwera,
    3. Serwer zapisuje do specjalnego pliku informację o przerwanej operacji. Zawiera się w niej między innymi nazwa użytkownika pobierającego plik, kod operacji oraz ostatni wysłany fragment pliku,
    4. W tym samym czasie klient zapisuje do pliku informację o przerwanej operacji. Zawiera się w niej między innymi lokalizacja pliku `.partial` i kod operacji,
    5. Klient i serwer oczekują na odzyskanie łączności z siecią,
    6. Klient nawiązuje ponownie połączenie z serwerem, loguje się na swoje konto,
    7. Klient sprawdza czy żadna operacja nie została wcześniej przerwana. Tak jest, zatem kieruje do użytkownika zapytanie o ponowienie operacji,
    8. Dwa możliwe scenariusze:
        - Użytkownik ponawia pobieranie pliku:
            (a) Aplikacja klienta wysyła prośbę o ponowienie operacji pobierania,
            (b) Serwer, po zweryfikowaniu że taka operacja faktycznie miała wcześniej miejsce, wyraża zgodę na ponowienie,
            (c) Kroki 3 – 8 scenariusza głównego.
        - Użytkownik kończy operację pobierania pliku:
            (a) Aplikacja klienta wysyła informację o tym, że pobieranie nie będzie ponawiane,
            (b) Serwer po otrzymaniu informacji usuwa wpis utworzony w punkcie 3 scenariusza alternatywnego IV.

2.13. Ładowanie plików

Klient może załadować plik na serwer do aktualnego katalogu:

- Scenariusz główny:
    1. Klient wysyła pytanie o zgodę na załadowanie pliku. Zawiera się w niej między innymi nazwa przesyłanego pliku oraz jego rozmiar,
    2. Serwer, po zweryfikowaniu uprawnień użytkownika, wydaje zgodę na ładowanie wraz z dodatkowymi informacjami (np. jaki jest maksymalny możliwy rodzaj pojedynczego pakietu, kod identyfikujący operację),

3. Klient nawiązuje dodatkowe połączenie TCP z serwerem, które będzie wykorzystywane do przesyłania danych,
4. Klient dzieli plik na małe pakiety i wysyła je do serwera,
5. Serwer odbiera fragmenty od klienta i dopisuje je do pliku o nazwie "`[NAZWA-ŁADOWANEGO-PLIKU].[SÓL].partial`" reprezentującego częściowo załadowane dane. Plik ten nie jest wyświetlany w przypadku żądania wypisania katalogu przez klienta,
6. Ostatnim pakietem wysłanym przez klienta jest informacja o zakończeniu ładowania,
7. Serwer zmienia nazwę pliku częściowego na docelową,
8. Utworzone wcześniej dodatkowe połączenie TCP nie jest zamykane od razu – istnieje możliwość wykorzystania go do kolejnych ładowań (lub pobrań).

- Scenariusz alternatywny I - odmowa ładowania:
  1. Krok 1 scenariusza głównego,
  2. Klient otrzymuje odmowę ładowania pliku – użytkownik dostaje informację o niepowodzeniu operacji.
- Scenariusz alternatywny II – dodatkowe połączenie TCP z serwerem zostało wcześniej zestawione:
  1. Kroki 1 – 2 scenariusza głównego,
  2. Dodatkowe połączenie już istnieje – serwer i klient będą je wykorzystywać do przesyłania danych,
  3. Kroki 4 – 8 scenariusza głównego.
- Scenariusz alternatywny III – dodatkowe połączenie TCP nie mogło zostać zestawione:
  1. Kroki 1 – 2 scenariusza głównego,
  2. Dodatkowe połączenie nie mogło zostać zestawione – użytkownik dostaje informację o niepowodzeniu operacji.
- Scenariusz alternatywny IV – klient wysłał nieprawidłowy fragment pliku:
  1. Kroki 1 – 4 scenariusza głównego,
  2. Klient wysłał nieprawidłowy fragment pliku – serwer wysyła informację o błędzie i kończy operację. Nie zamyka on jednak zestawionego połączenia TCP – może być wykorzystane później.
- **Scenariusz alternatywny V – przerwanie połączenia z siecią w trakcie ładowania pliku**:
  1. Kroki 1 – 4 scenariusza głównego,
  2. Przerwanie połączenia ze strony klienta lub serwera,
  3. Serwer zapisuje do specjalnego pliku informację o przerwanej operacji. Zawiera się w niej między innymi nazwa użytkownika ładującego plik, kod operacji oraz ostatni odebrany fragment pliku,

4. W tym samym czasie klient zapisuje do pliku informację o ostatnim wysłanym fragmencie pliku oraz lokalnej ścieżce do pliku,
5. Klient i serwer oczekują na odzyskanie łączności z siecią,
6. Klient nawiązuje ponownie połączenie z serwerem, loguje się na swoje konto,
7. Klient sprawdza czy żadna operacja nie została wcześniej przerwana. Tak jest, zatem kieruje do użytkownika zapytanie o ponowienie operacji,
8. Dwa możliwe scenariusze:
   - Użytkownik ponawia ładowanie pliku:
     (a) Aplikacja klienta wysyła prośbę o ponowienie operacji ładowania,
     (b) Serwer, po zweryfikowaniu że taka operacja faktycznie miała wcześniej miejsce, wyraża zgodę na ponowienie,
     (c) Kroki 3 – 8 scenariusza głównego.
   - Użytkownik kończy operację ładowania pliku:
     (a) Aplikacja klienta wysyła informację o tym, że ładowanie nie będzie ponawiane,
     (b) Serwer po otrzymaniu informacji usuwa wpis utworzony w punkcie 3 scenariusza alternatywnego V.

2.14. Struktura komunikatów

2.14.1 Polecenia dla serwera

Polecenia w protokole TDP wysyłane są tekstowo, podobnie jak np. w protokole HTTP. W pierwszej linii komunikatu znajduje nazwa polecenia, natomiast w kolejnych jego parametry.

Poniższa tabela przedstawia nazwy poleceń w protokole TDP:

Polecenie	**Opis**	**Parametry**
`auth`	Autoryzacja	Login i hasło
`cd`	Zmiana katalogu	Ścieżka do zmiany
`pwd`	Wyświetlanie aktualnego katalogu	—
`ls`	Wypisywanie katalogów	Parametry określające prośbę o zwrot dodatkowych informacji
`tree`	Wypisywanie drzewa katalogów	—
`mkdi`	Tworzenie katalogu	Nazwa nowego katalogu
`rm`	Usuwanie pliku lub katalogu	Nazwa pliku lub katalogu
`rname`	Zmiana nazwy pliku lub katalogu	Nazwa pliku oraz nowa nazwa
`cp`	Kopiowanie pliku lub katalogu	Nazwa pliku oraz miejsce docelowe
`mv`	Przenoszenie pliku lub katalogu	Nazwa pliku oraz miejsce docelowe
`dl`	Inicjacja pobierania	Nazwa pliku
`ul`	Inicjacja ładowania	Nazwa pliku oraz jego rozmiar

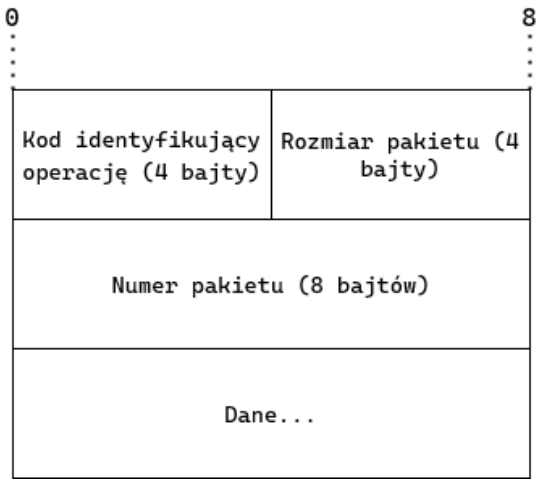
2.14.2 Odpowiedzi serwera na polecenia

W odpowiedzi serwera w kolejnych liniach zawierają się następujące informacje:

1. Kod zwrotny – określa np. status powodzenia operacji, informację o ewentualnym błędzie,
2. Nazwa polecenia – nazwa polecenia wykonanego przez użytkownika,
3. Szczegóły odpowiedzi – rezultat wykonanego polecenia. Przykładowo, dla polecenia `ls` w odpowiedzi w kolejnych liniach zawarte będą nazwy plików i katalogów wraz z np. datami ich utworzenia.

2.14.3 Transmisja danych

Przy pobieraniu lub ładowaniu plików na serwer dane przesyłane są w małych pakietach w sposób binarny. Poniższy rysunek przedstawia strukturę takiego binarnego pakietu:



Rysunek 3: Pakiet używany przy pobieraniu lub ładowaniu

3. Serwer

3.1. Słownik pojęć

Pojęcia używane w tej sekcji:

1. TDS – TinDox Server.
2. Instancja serwera – katalog zawierający katalog podrzędny o nazwie `.tds`, w którym znajdują pliki konfiguracyjne potrzebne do uruchomienia serwera. Instancja jest również korzeniem systemu plików widocznego przez klienta.
3. IO - wejście i wyjście.

3.2. Wymagania funkcjonalne

1. Możliwość szybkiego utworzenia instancji serwera w dowolnym miejscu na dysku.
2. Tworzenie, modyfikacja i usuwanie użytkowników uprawnionych do korzystania z dysku sieciowego.
3. Realizacja komunikacji z klientami z wykorzystaniem gniazd BSD.
4. Wydajne IO dzięki wykorzystaniu zasobów systemowych:
   - Wykorzystanie efektywnej liczby wątków do obsługi wielu klientów jednocześnie,
   - Wykorzystanie linuksowego mechanizmu `epoll` do sprawnej obsługi aktywnych klientów.
5. Bezbłędne zamykanie połączeń z klientami oraz całego serwera.
6. Prawidłowe reagowanie na sygnały systemowe (np. użycie `Ctrl+C` w terminalu powinno prawidłowo zakończyć działanie serwera).
7. Monitorowanie połączeń przychodzących oraz błędów serwera, tworzenie raportów i zarządzanie nimi.

3.3. Wymagania niefunkcjonalne

1. Konfigurowalność – serwer będzie wykorzystywał plik konfiguracyjny w formacie TOML. Edytując go, użytkownik będzie mógł dostosować serwer do możliwości swojego komputera, co oznacza między innymi:
   - Możliwość wyboru maksymalnej ilości wątków używanych przez serwer (domyślnie będzie to wartość zwracana przez funkcję z języka `C++` – `std::thread::hardware_concurrency()`),
   - Możliwość wyboru maksymalnej ilości klientów w sesji.
2. Wydajność – serwer powinien sprawnie obsługiwać wiele żądań jednocześnie przy wykorzystaniu optymalnej ilości zasobów systemu operacyjnego.
3. Bezpieczeństwo – serwer powinien być odporny na złośliwe zapytania. Oznacza to, że w aplikacji nie wystąpią podatności takie jak np. RCE.

3.4. Polecenia linii komend

Przykładowe polecenia linii komend udostępniane przez serwer:

- `./tds init` – tworzenie instancji serwera w bieżącym katalogu.
- `./tds run <flags>` – uruchomienie instancji serwera w bieżącym katalogu.
- `./tds log <flags>` – przeglądanie logów serwera.
- `./tds user <flags>` – manipulacja listą użytkowników (np. dodawanie nowego użytkownika "`./tds user add ...`").
- `./tds config <flags>` – zmiana wartości parametrów serwera.

3.5. Narzędzia i biblioteki

Element	Narzędzia	Wersja
Język programowania	`C++`	ISO/IEC 14882:2020
Kompilator	`g++` `clang`	11.1.0 13.0.0
System budowania	`CMake`	3.18.4
Automatyzacja testów	`CTest`	3.18.4
Testy jednostkowe	`Catch2`	3.0.0
Obsługa formatu TOML	`@marzer/tomlplusplus`	2.5.0
Platforma docelowa	`Linux x86-64`	4.0.0

4. Klient mobilny

4.1. Opis klienta

Połączenie z serwerem za pomocą klienta w aplikacji mobilnej zaimplementowane z użyciem języka kotlin w środowisku Android Studio. Za ich pomocą powstanie prosta aplikacja mobilna z interakcyjnym interfejsem graficznym reprezentująca nasz zdalny system plików. Aplikacja będzie budowana na system Android.

4.2. Narzędzia i biblioteki

Element	Narzędzia
Język programowania	`Kotlin`
Kompilator	`kotlinc`
IDE	`Android Studio`
Platforma docelowa	`Android`

5. Klient okienkowy

5.1. Opis klienta

Połączenie z serwerem realizowane przez klienta okienkowego zaimplementowanego z użyciem języka Java i klasy Socket, która reprezentuje gniazda klienckie. Po uruchomieniu klient podejmie próbę połączenia z serwerem. W przypadku udanego połączenia aplikacja umożliwi interakcję ze zdalnym systemem plików.

5.2. Narzędzia i biblioteki

Element	Narzędzia
Język programowania	`Java`
Kompilator	`javac`
Testy jednostkowe	`JUnit`
System budowania	`Gradle`

6. Klient konsolowy

6.1. Opis klienta

Połączenie z serwerem realizowane poprzez klienta zaimplementowanego przy użyciu biblioteki `ncurses` i języka `C`. Ncurses to zbiór funkcji i narzędzi pozwalających tworzyć zgrabne TUI. Wykorzystując elementy biblioteki powstanie narzędzie do wykonywania operacji plikowych na zdalnym systemie plików.

Klient konsolowy przeznaczony jest na system `Ubuntu`. Będzie kompilowany z wykorzystaniem `gcc` oraz budowany z wykorzystaniem `CMake`.

6.2. Narzędzia i biblioteki

Element	Narzędzia	Wersja
Język programowania	`C`	ISO/IEC 9899:2011
Kompilator	`gcc`	10.3.0
System budowania	`CMake`	3.22.0
TUI	`ncurses`	6.2
Platforma docelowa	`Linux x86-64`	4.0.0

7. Metody testowania

1. Testy jednostkowe – testowanie najważniejszych modułów poszczególnych projektów (np. modułu raportującego w serwerze lub składowych konstrukcji MVC w aplikacji mobilnej).

2. Testy integracyjne – testy weryfikujące poprawność komunikacji między największymi składnikami całego systemu (np. między klientem konsolowym a serwerem).

3. *Fuzzing* – automatyczne testowanie reakcji serwera na losowe wiadomości przychodzące od klienta.

4. Testy obciążeniowe – zautomatyzowane testowanie wytrzymałości serwera na zapytania przychodzące od dużych ilości klientów.

5. Testy penetracyjne – testy wykonywane w celu znalezienia podatności w oprogramowaniu serwera oraz klientów.

6. Testy empiryczne – testy wykonywane w czasie prezentacji programów dowodzące poprawności ich działania.

8. Zespół

8.1. Środowisko deweloperskie

Element	**Narzędzia**	**Wersja**
System operacyjny	`Ubuntu`	20.04, 21.10
	`Manjaro Linux`	21.1.6
	`Windows`	10.0
	`Arch Linux`	—
Pomocniczy język skryptowy[1]	`Python`	3.9.7
	`Bash`	5.1.8
Kontrola wersji	`git`	2.32.0
Repozytorium	`GitHub`	—
ITS[2]	`GitHub Issues`	
Tablica kanban	`GitHub Projects`	
CI/CD	`Github Actions`	—
Dokumentacja	`Overleaf`	—

8.2. Podział pracy

Projekt	**Wykonawca**
Serwer	Jakub Mazurkiewicz
Klient mobilny	Damian Piotrowski
Klient okienkowy	Anna Pyrka
Klient konsolowy	Łukasz Reszka

[1] Języki skryptowe będą używane do np. symulowania złożonych przypadków testowych, automatyzacji testów.

[2] Issue tracking system.