# Sieci Komputerowe
# laboratorium - projekt

Zuzanna Juszczak 141238
Stanisław Kaczmarek 141240

## 1 Temat projektu

Wybrany przez nas temat to "System robota indeksującego strony HTML wraz z aplikacją umożliwiającą wyszukiwanie stron na podstawie słów kluczowych".

## 2 Opis protokołu komunikacyjnego

### 2.1 Zapytanie

Po otwarciu gniazda klient wysyła zapytanie do serwera - łańcuch znaków z którego odczytane zostaną słowa do wyszukania. Wiadomość taka zakodowana jest jako:

- długość - 4-bajtowa liczba,
- treść zapytania - tablica bajtów o wcześniej odczytanej długości.

### 2.2 Odpowiedź

Na odpowiedź serwera składają się słowa (klient może chcieć zweryfikować jakie słowa serwer odczytał z jego zapytania), oraz lista adresów URL, na których znaleziono wszystkie wyszukiwane słowa.

Wiadomośc taka zakodowana jest jako:

- liczba słów - 4-bajtowa liczba,
- zbiór słów jako:
  - długość słowa - 4-bajtowa liczba
  - słowo o odczytanej wcześniej ilości bajtów
- liczba adresów - 4-bajtowa liczba,
- po tej liczbie znajduje się zbiór adresów, jako:
  - długość adresu - 4-bajtowa liczba,
  - adres o odczytana wcześniej ilość bajtów

# 3 Opis implementacji oraz opis plików źródłowych

- server
  - main.cc - główny plik serwera, odczytujący argumenty wywołania i tworzący odpowiednie klasy
  - loader
    - loader.h - plik nagłówkowy klasy Loader, zajmującej się odczytywaniem stron ze znalezionych linków (odczytywanie strony w klasie Website) i umieszczaniem słów i odpowiadających im adresów w klasie Storage.
    - loader.cc - implementacja klasy Loader
  - server
    - client_data.h - struktura ClientData, stosowana jako argument wątku obsługującego połączenie między klientem a serwerem
    - server.h - plik nagłówkowy klasy Server, odpowiedzialnej za działanie serwera (choć do samego przesyłania komunikatów są osobne klasy)
    - server.cc - implementacja klasy Server
    - server_request.h - plik nagłówkowy klasy ServerRequest, definiującej format zapytania klienta do serwera
    - server_request.cc - implementacja klasy ServerRequest
    - server_response.h - plik nagłówkowy klasy ServerResponse, definiujcej format odpowiedzi serwera do klienta
    - server_response.cc - implementacja klasy ServerResponse
  - storage
    - storage.h - plik nagłówkowy klasy Storage, zawierającej poindeksowane słowa, pozwalająca klasie Loader zapisywać kolejne słowa, a klasie Server odczytywać adresy odpowiadające danemu słowu
    - storage.cc - implementacja klasy Storage
  - util
    - coding.h - zawiera funkcje pozwalające zapisywać i odczytywać liczby na podstawie tablicy bajtów
    - options.h - struktura Options inicjalizowana na podstawie parametrów wywołania i służąca do inicjalizacji niektórych klas
    - record.h - klasa Record zawierająca adres i liczbę wystąpień słowa oraz odpowiadający jest komparator, co pozwala klasie Storage przechowywać takie rekordy w std::set posortowane malejąco po liczbie wystąpień i alfabetycznie po adresach
    - word_splitter.h - plik nagłówkowy klasy WordSplitter służącej podziale stringa na pojedyncze słowa
    - word_splitter.cc - implementacja klasy WordSplitter
  - website
    - get_request.h - plik nagłówkowy klasy GetRequest dokonującej zapytania HTTP typu GET, w celu pobrania treści strony internetowej
    - get_request.cc - implementacja klasy GetRequest
    - website.h - plik nagłówkowy klasy Website, odpowiedzialnej za zliczanie słów na pojedynczej stronie i odnajdywanie znajdujących się na niej linków
    - website.cc - implementacja klasy Website

  - Makefile - plik Makefile

- client
  - src/main/java/put/sk/pseudogoogle
    - application/Application.java - główna klasa klienta
    - components/frames
      - CommunicationFrame.java - klasa odpowiadająca za okno wysyłające zapytanie do serwera
      - ConnectionFrame.java - klasa odpowiadająca za okno do połączenia się z serwerem
      - ResponseFrame.java - klasa odpowiadająca za okno wyświetlające odpowiedź otrzymaną od serwera - listę stron, na których znaleziono podane słowo
      - ReconnectionFrame.java - klasa odpowiadająca za okno pokazujące próbę ponownego połączenia z tym samym serwerem
    - data
      - Response.java - klasa zawierająca odpowiedź serwera
    - error
      - ConnectionException.java, InvalidAddressException.java, InvalidPortException.java - klasy wyjątków obsługiwanych z poziomu interfejsu użytkownika
    - logic
      - communication/Communicator.java - klasa odpowiedzialna za komunikację między klientem a serwerem - wysyłanie zapytania i odbieranie odpowiedzi
      - connection/Connector.java - klasa odpowiedzialna za nawiązywanie połączenia między klientem, a serwerem
  - pom.xml - plik pom.xml

# 4 Sposób kompilacji, uruchomienia i obsługi programu

## 4.1 Kod źródłowy

Kod źródłowy można pobrać wywołując:

```
$ git clone https://gitlab.cs.put.poznan.pl/inf141240/pseudogoogle.git
```

## 4.2 Serwer

Aplikacja serwera napisana jest w języku C++, a do kompilacji wykorzystuje skrypt Makefile. Do poprawnego działania potrzebuje 2 bibliotek zewnętrznych:

- libcurl
- gumbo

W systemie Ubuntu można je pobrać, wykonując polecenia:

```
$ sudo apt install libcurl4-openssl-dev
$ sudo apt install libgumbo-dev
```

### 4.2.1 Kompilacja

Kod klienta znajduje się w podkatalogu client. Jego kompilacji można dokonać wykonując w tym katalogu polecenie:

```
$ make
```

### 4.2.2 Uruchomienie

Wynikiem kompilacji jest plik pseudogoogle_server w podkatalogu server. Uruchomienie go bez parametrów lub z parametrem –help skutkuje pojawieniem się informacji o innych parametrach wywołania.

Adres pierwszej strony do odwiedzenia podaje się w parametrze –url=<string>, np.

```
$ ./pseudogoogle_server --url=https://example.com
```

Dodatkowo istnieją jeszcze parametry:

- –max-depth=<int> decydujący a maksymalnej wysokości drzewa powstałego przy przeszukiwaniu znalezionych linków, domyślnie 5
- –node-limit=<int> określający ewentualne ograniczenie na ilość wierzchołków w tym drzewie, domyślnie 100

Wartość -1 przy tych parametrach powoduje wyłączenie danego ograniczenia, przy czym nie mogą oba być równe -1.

### 4.2.3 Działanie

Aplikacja serwera po uruchomieniu najpierw indeksuje strony, a potem uruchamia właściwy serwer, pozwalając na komunikację ze strony klienta.

## 4.3 Klient

Aplikacja klienta napisana jest w języku Java, a do kompilacji wykorzystuje narzędzie Maven.

### 4.3.1 Kompilacja

Aby skompilować klienta, należy w podkatalogu client uruchomić polecenie:

```
$ mvn install
```

### 4.3.2 Uruchomienie

Wynikiem kompilacji jest plik pseudogoogle-client-jar-with-dependencies.jar w katalogu client/target. Uruchomić go można poleceniem:

```
$ java -jar pseudogoogle-client-jar-with-dependencies.jar
```

### 4.3.3 Działanie

Aplikacja klienta działa przełączając się między 3 interaktywnymi oknami.

Pierwsze z nich oczekuje od użytkownika podania adresu i portu serwera. Zamknięcie tego okna powoduje zamknięcie aplikacji.

Po poprawnym nawiązaniu połączenia, pojawia się okno pozwalające przesłać zapytanie o słowa, których użytkownik szuka. Zamknięcie tego okna prowadzi z powrotem do okna nawiązywania połączenia.

Po bezproblemowym odebraniu odpowiedzi, pojawia się trzecie okno wyświetlające wynik zapytania, a więc, przede wszystkim, listę znalezionych adresów URL. Po zamknięciu okna z odpowiedzią aplikacja próbuje nawiązać nowe połączenie z tym samym serwerem (poprzednie zostaje zamknięte po otrzymaniu odpowiedzi). Jeśli jej się uda przełącza się na okno z zapytaniem, jeśli nie - na to do nawiązywania połączenia.