



**AKADEMIA GÓRNICZO-HUTNICZA**

Dokumentacja do projektu

## **Prosta lista zadań**

z przedmiotu

### **Programowanie Sieciowe**

Elektronika i Telekomunikacja 3 rok

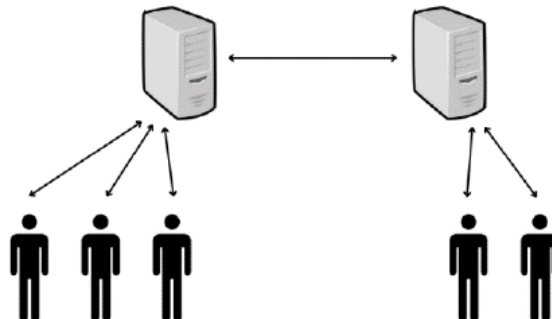
*Jakub Kostecki*  
*Szymon Krzyworzeka*  
*Marcel Włosek*

prowadzący: mgr. inż. Karol Salwik

22.01.2023

# 1. Opis projektu

Tematem zrealizowanego projektu było stworzenie komunikatora tekstowego typu klient-serwer, wykorzystując dwa skomunikowane ze sobą serwery. Klienci mają możliwość podłączenia się do jednego z serwerów i tym samym dołączenia się do usługi „chatu”. Wiadomość wysłana przez klienta do serwera do którego jest podłączony jest rozsyłana do wszystkich klientów podłączonych do tego samego serwera jak również do wszystkich klientów połączonych do drugiego serwera.



Rysunek 1- diagram architektury komunikatora

## 2. Realizacja projektu

Projekt został wykonany przy użyciu języka Python3. W oparciu o funkcje sieciowe stworzono dwa programy. Program serwera działający, jednocześnie do komunikacji z klientami i drugim serwerem, oraz program klienta pozwalający na połączenie się z serwerem. Wykorzystano protokół komunikacyjny TCP – Transmission Control Protocol, oraz adresację Ipv4 unicast. Wykorzystanie biblioteki threading umożliwiło realizowanie każdego połączenia w obrębie osobnego wątku. Działanie programów wymagało umożliwienia wielu procesom pracowania jednocześnie

Kodowanie wiadomości będzie wykonywane w programie w sposób następujący: Proces wysyła dwie zakodowane wiadomości z czego pierwsza to strumień bajtów odpowiadający długości wysyłanej wiadomości właściwej uzupełniony przestrzenią bitową do długości 64. Druga wiadomość to już właściwa zakodowana wiadomość pobrana z klawiatury klienta.

## 3. Prezentacja działania programu

Prezentacja została wykonana na przykładzie połączenia jednego klienta do pierwszego serwera, dwóch klientów do drugiego serwera, oraz nawiązaniu połączenia pomiędzy serwerami.

Pierwsza maszyna wirtualna

Konsole serwera o adresie .101 i klienta podłączony do serwera .102:

```
ubuntu31@ubuntu31-VirtualBox:~/PS/projekt_fln$ python3 server.py
Server is starting
[Listening] server is listening on 192.168.56.101
accepted connection from ('192.168.56.102', 55198)
[Active Connections] 1.0
accepted connection from ('192.168.56.102', 55214)
[Active Connections] 2.0
accepted connection from ('192.168.56.104', 41842)
[Active Connections] 3.0
dzien dobry

ubuntu31@ubuntu31-VirtualBox:~/PS/projekt_fln$ python3 klient.py
dzien dobry
dzien dobry
```

Druga maszyna wirtualna

Konsole serwera o adresie .102 i klienta podłączony do serwera .101:

```
ubutu32@ubutu32-VirtualBox:~/PS/projekt_fin$ python3 serwer.py
Server is starting
[Listening] server is listening on 192.168.56.102
accepted connection from ('192.168.56.101', 60300)
[Active Connections] 1.0
accepted connection from ('192.168.56.101', 60306)
[Active Connections] 2.0
dzien dobry

ubutu32@ubutu32-VirtualBox:~/PS/projekt_fin$ python3 klient.py
dzien dobry
```

Trzecia maszyna wirtualna

Konsola klienta podłączonego do serwera o adresie .101

```
ubuntu31@ubuntu31-VirtualBox:~/PS/projekt_fin$ python3 klient.py
dzien dobry
```

Po wysłaniu wiadomości z klienta podłączonego do serwera .102, została ona przekazana do wszystkich klientów wliczając w to nadawcę.

Stany portów na pierwszej maszynie wirtualnej:

```
ubuntu31@ubuntu31-VirtualBox:~$ netstat -na | grep 2000
tcp        0      0 192.168.56.101:2000 0.0.0.0:*        LISTEN
tcp        0      0 192.168.56.101:2000 192.168.56.102:55198 ESTABLISHED
tcp        0      0 192.168.56.101:2000 192.168.56.104:41842 ESTABLISHED
tcp        0      0 192.168.56.101:2000 192.168.56.102:55214 ESTABLISHED
ubuntu31@ubuntu31-VirtualBox:~$ netstat -na | grep 2001
tcp        0      0 192.168.56.101:60306 192.168.56.102:2001 ESTABLISHED
tcp        0      0 192.168.56.101:60300 192.168.56.102:2001 ESTABLISHED
```

Stany portów na drugiej maszynie wirtualnej:

```
ubutu32@ubutu32-VirtualBox:~$ netstat -na | grep 2001
tcp        0      0 192.168.56.102:2001 0.0.0.0:*        LISTEN
tcp        0      0 192.168.56.102:2001 192.168.56.101:60306 ESTABLISHED
tcp        0      0 192.168.56.102:2001 192.168.56.101:60300 ESTABLISHED
ubutu32@ubutu32-VirtualBox:~$ netstat -na | grep 2000
tcp        0      0 192.168.56.102:55214 192.168.56.101:2000 ESTABLISHED
tcp        0      0 192.168.56.102:55198 192.168.56.101:2000 ESTABLISHED
```

Przepływ pakietów na drugiej maszynie podczas wysłania komunikatu:

```
ubutu32@ubutu32-VirtualBox:~$ sudo tcpdump -i enp0s3 -n port 2001
[sudo] password for ubutu32:
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), snapshot length 262144 bytes
20:12:22.183728 IP 192.168.56.101.60300 > 192.168.56.102.2001: Flags [P.], seq 4228016093:4228016157, ack 3125892060, win 502, options [nop,nop,TS val 1998942503 ecr 1833721960], length 64
20:12:22.224746 IP 192.168.56.102.2001 > 192.168.56.101.60300: Flags [.], ack 64, win 509, options [nop,nop,TS val 1833721956 ecr 1998942503], length 0
20:12:22.225056 IP 192.168.56.101.60300 > 192.168.56.102.2001: Flags [P.], seq 64:67, ack 1, win 502, options [nop,nop,TS val 1998942544 ecr 1833721956], length 3
20:12:22.225064 IP 192.168.56.102.2001 > 192.168.56.101.60300: Flags [.], ack 67, win 509, options [nop,nop,TS val 1833721956 ecr 1998942544], length 0
20:12:22.228851 IP 192.168.56.102.2001 > 192.168.56.101.60306: Flags [P.], seq 2353311543:2353311546, ack 1525762347, win 510, options [nop,nop,TS val 1833721960 ecr 1997837707], length 3
20:12:22.228885 IP 192.168.56.102.2001 > 192.168.56.101.60300: Flags [P.], seq 1:4, ack 67, win 509, options [nop,nop,TS val 1833721960 ecr 1998942544], length 3
20:12:22.229110 IP 192.168.56.101.60300 > 192.168.56.102.2001: Flags [.], ack 4, win 502, options [nop,nop,TS val 1998942549 ecr 1833721960], length 0
20:12:22.272540 IP 192.168.56.101.60306 > 192.168.56.102.2001: Flags [P.], ack 3, win 502, options [nop,nop,TS val 1998942592 ecr 1833721960], length 0
20:12:22.285964 IP 192.168.56.101.60306 > 192.168.56.102.2001: Flags [P.], seq 1:65, ack 3, win 502, options [nop,nop,TS val 1998942605 ecr 1833721960], length 64
20:12:22.286030 IP 192.168.56.102.2001 > 192.168.56.101.60306: Flags [.], ack 65, win 510, options [nop,nop,TS val 1833722017 ecr 1998942605], length 0
20:12:22.286216 IP 192.168.56.101.60306 > 192.168.56.102.2001: Flags [P.], seq 65:68, ack 3, win 502, options [nop,nop,TS val 1998942606 ecr 1833722017], length 3
20:12:22.286221 IP 192.168.56.102.2001 > 192.168.56.101.60306: Flags [.], ack 68, win 510, options [nop,nop,TS val 1833722017 ecr 1998942606], length 0
```

## 4. Wykorzystane bibliotek i funkcje

- Biblioteka socket

1. Funkcja `socket()` – tworzenie gniazd w programie serwera i klienta
2. Funkcja `gethostbyname()` – pobieranie adresu IP
3. Funkcja `gethostname()` – pobieranie nazwy urządzenia
4. Funkcja `bind()` – powiązanie z gniazdem
5. Funkcja `connect()` – nawiązywanie połączenia
6. Funkcja `listen()` – nasłuchiwanie na serwerze
7. Funkcja `accept()` – akceptacja połączeń przez serwer
8. Funkcja `send()` – wysyłanie informacji
9. Funkcja `recv()` – odbieranie informacji

- Biblioteka threading

Funkcja `decode()` – umożliwiająca zamienienie stringa bitowego na stringa w kodzie Unicode

Funkcja `encode()` – umożliwiająca zakodowanie zmiennej string jako strumienia bitowego

Funkcja `Thread()` – umożliwia wielowątkowość

## 5. Napotkane problemy i ich rozwiązania

Problemem okazała się implementacja zakładanego w fazie konspektu protokołu SCTP. Mimo usilnej próby postawienia gniazda `IPPROTO_SCTP` lub `SOCK_SEQPACK` nie udało się znaleźć sposobu, aby kompilator nie wyrzucał błędu z nieobsługiwaną rodziną adresową. Zdecydowano się zatem zrealizować połączenie między serwerami za pomocą gniazd strumieniowych.

Podczas pisania funkcjonalności komunikacji serwera z serwerem, ponieważ serwery obsługują się nawzajem tak samo jak obsługują klientów. Napotkaliśmy na problem w którym serwer odbierający wiadomość od drugiego serwera wysyła ją do wszystkich połączonych ze sobą klientów. Ponieważ w liście klientów znajduje się też serwer od którego otrzymano wiadomość, wiadomość jest wysyłana z powrotem. Sposobem na rozwiązanie takiego problemu okazało się być wprowadzenie zmiennej, która trzyma treść poprzednio otrzymanej wiadomości. Nie jest to rozwiązanie idealne z dwóch powodów: Nie jest możliwe wysłanie wiadomości o takiej samej treści dwa razy z rzędu oraz serwer wysyła wiadomość o jeden raz za dużo np. w przypadku, gdy klient połączony z serwerem o adresie `x.x.x.12` wyśle wiadomość i serwer `x.x.x.12` wyśle tę wiadomość do serwera `x.x.x.34` to serwer `x.x.x.34` wysyła tę wiadomość z powrotem, mimo że serwer `x.x.x.12` ją zignoruje.

## 6. Dalszy rozwój i ulepszenia

Implementacja mechanizmów zabezpieczających takich jak, bezpieczne rozłączanie klientów w momencie kiedy serwer nagle przestanie działać.

Na ten moment klient może bezpiecznie rozłączyć się z serwerem jedynie poprzez wysłanie wiadomości o treści `DISCONNECT`, nie jest to jednak wygodne

Wdrożenie interfejsu użytkownika po stronie klienta. Możliwość wybrania pseudonimu który będzie wyświetlany podczas wysyłania wiadomości na komunikatorze.