UNIWERSYTET RZESZOWSKI

WYDZIAŁ NAUK ŚCISŁYCH I TECHNICZNYCH

INSTYTUT INFORMATYKI

Krystian Filipek

134907

Informatyka

Komunikator „Koliber”

Praca projektowa

Praca wykonana pod kierunkiem

mgr inż. Jarosław Szkoła

Rzeszów 2026

Spis treści

1. Wprowadzenie

Projekt prostego, anonimowego komunikatora sieciowego został zrealizowany na potrzeby przedmiotu **Sieci Komputerowe** na 2 roku Informatyki na **Uniwersytecie Rzeszowskim**. Głównym celem pracy było zaprojektowanie i implementacja systemu wielowątkowego wymiany informacji w architekturze **Klient-Serwer-Klient**. System pozwala na równoczesną komunikację wielu jednostek klienckich za pośrednictwem centralnego węzła pośredniczącego, którym w naszym przypadku jest serwer **Ubuntu**. Zarządza on przepływem danych w izolowanym środowisku sieciowym na potrzeby symulacji realnej sieci.

The project of a simple, anonymous network messenger was carried out for the purposes of the **Computer Networks** course in the second year of Computer Science at the **University of Rzeszów**. The main goal of the work was to design and implement a multithreaded information exchange system in the **Client-Server-Client** architecture. The system allows for simultaneous communication between multiple client units via a central intermediary node, which in our case is an Ubuntu server. It manages data flow in an isolated network environment for the purposes of simulating a real network.

2. Opis założeń projektu

Celem projektu było stworzenie prostego i anonimowego komunikatora sieciowego przy użyciu języka programowania **Python**. Program ma za zadanie umożliwić wielu użytkownikom jednoczesną wymianę wiadomości tekstowych w czasie rzeczywistym, wykorzystując arhitekturę wielowątkową również zaimplementowaną używając wspomnianego wcześniej języka **Python** dobierając wcześniej odpowiednie moduły takie jak `threading` czy `socket`. Program ma działać w środowisku rozproszonym, łącząc maszyny o różnych systemach operacyjnych pracując wewnątrz wirtualnej infrastruktury sieciowej.

Podstawowym problemem, który zostanie rozwiązany przez realizację tego projektu, jest zagrożenie prywatności w sieci oraz konieczność udostępniania danych osobowych w tradycyjnych systemach komunikacji. Obecnie wiele krajów zachodnich takich jak chociażby *Wielka Brytania* coraz częściej z wrogością zwraca się do prywatnej wymiany informacji w obawie o to iż użytkownicy będą komunikować między sobą informacje nie będące po drodze narracji rządowej. Projekt inspirując się forum obrazkowym *4chan* rozwiązuje ten problem poprzez brak konieczności zakładania kont oraz wykorzystanie tymczasowych identyfikatorów sesji, które są generowane losowo przez serwer i usuwane natychmiast po rozłączeniu użytkownika z pamięci operacyjnej węzła.

Aby problem został skutecznie rozwiązany, potencjalny zespół musi posiadać wiedzę z zakresu architektury sieciowej **TCP/IP**, modelu **ISO/OSI** (w szczególności warstwy transportowej i aplikacji) oraz podstawowej obsługi wątków.

Rozwiązanie problemu przebiegło w kilku zdefiniowanych krokach. W pierwszej kolejności nastąpiło zaprojektowanie topologii sieciowej w środowisku `VirtualBox`. Kolejnym krokiem było zaimplementowanie logicznej warstwy serwera, odpowiedzialnego za akceptowanie połączeń i przesyłanie komunikatów. Następnie wdrożono obsługę wielu wątków oraz system anonimowego przydzielania identyfikatorów. Końcowym etapem było stworzenie podstawowego interfejsu **CLI** dla klientów, gdzie Ci będą mogli odróżniać wiadomości swoje od cudzych za pośrednictwem kolorów. Na samym końcu dodano również możliwość przywrócenia starej konwersacji poprzez wpisanie swojego starego ID użytego wcześniej.

2.1. Wymagania funkcjonalne

- **Nawiązywanie połączenia:** Użytkownik musi mieć możliwość połączenia się z serwerem za pomocą protokołu TCP/IP po podaniu adresu IP serwera oraz zdefiniowanego portu (55555) - poziom skryptu `client.py`.
- **Automatyczne przydzielenie tożsamości:** System musi wygenerować i przypisać każdemu użytkownikowi unikalny, 6-znakowy identyfikator sesji - ID natychmiast po nawiązaniu połączenia przez użytkownika.
- **Wymaiana wiadomości w czasie rzeczywistym:** Wiadomość wpisana przez jednego użytkownika musi zostać natychmiast rozesłana przez serwer do wszystkich pozostałych aktywnych uczestników sesji.
- **Identyfikacja nadawcy:** Każda wiadomość przesłana w sieci musi być opatrzona identyfikatorem ID nadawcy, aby odbiorcy wiedzielo, od kogo pochodzi wiadomość.
- **Powiadomienia systemowe:** Program musi informować wszystkich uczestników o zmianach w strukturze sieci (dołączenia nowego użytkownika albo opuszczenie przez niego sesji).

- **Wizualne odróżnienie komunikatów:** Interfejs musi za pomocą kolorów odróżniać własne wiadomości od wiadomości innych użytokwników oraz komunikatów systemowych.
- **Automatyczne sprzątanie sesji:** Serwer musi automatycznie wykrywać rozłączanie klienta, zamykać powiązany z nim socket i zwalniać zajęte ID do ponownej puli losowania.
- **Przywrócenie konwersacji:** Gdy użytkownik posiada swoje stare ID, może przywrócić treść konwersacji w której uczestniczył.

2.2. Wymagania niefunkcjonalne

- **Niezawodność transmisji:** Komunikacja musi opierać się na protokole TCP, aby zagwarantować, że wiadomości dotrą do odbiorców w całości i w poprawnej kolejności.
- **Wielowątkowość:** Serwer musi być zdolny do obsługi wielu gniazd sieciowych (socketów) jednocześnie bez blokowania głownego procesu aplikacji.
- **Wieloplatformowość:** Kod musi być kompatybliny z różnymi systemami operacyjnymi począwszy od `Linux` aż po sysetmy z rodziny `Microsoft Windows`.
- **Minimalizm:** Oprogramowanie ma działać wyłącznie w trybie tekstowym CLI minimalizując tym samym zużycie pamięci RAM i CPU.
- **Brak śladu cyfrowego:** Wszystkie dane o użytkownikach i treści wiadomości muszą być przechowywane wyłącznie w pamięci RAM. Brak logowania danych na dysku twardym serwera zapewnia anonimowość i prywatność.
- **Izolacja środowiska:** System musi poprawnie pracować wewnątrz wirtualnej sieci (VirtualBox).
- **Samowystarczalność:** Projekt musi być napisany w czystym języku **Python** przy użyciu wyłącznie bibliotek standardowych, co elimunuje konieczność instalowania zewnętrznych bibliotek i zależności.

3. Opis struktury projektu

3.1. Struktura plików

Projekt został zorganizowany w sposób modułowy, oddzielając kod źródłowy od plików dokumentacji. Poniżej przedstawiona została struktura katalogów w formie drzewa:

```
koliber/
├── documentation/ ..... Dokumentacja techniczna projektu
├── src/ ..... Katalog z kodem źródłowym
│   ├── client.py ..... Aplikacja kliencka
│   └── server.py ..... Skrypt serwera
```

3.2. Najważniejsze metody

W tej sekcji omówiono kluczowe fragmenty implementacji odpowiedzialne za logikę sieciową, persystencję danych oraz wielowątkową obsługę użytkowników.

3.2.1. Plik server.py

Listing 3.1. Logika przekaźnika rozsyłająca wiadomości

```
def handle_broadcast(message_bytes, sender_socket):
    save_to_history(message_bytes.decode('utf-8'))

    for user_socket in user_sockets[:]:
        if user_socket != sender_socket:
            try:
                user_socket.send(message_bytes)
            except Exception as e:
                print("ERROR: rozsylanie wiadomosci" + str(e))
                if user_socket in user_sockets:
                    user_sockets.remove(user_socket)
```

Listing 3.2. Zarządzanie sesją oraz mechanizm przywracania historii

```
def handle_client(user, client_address):
    try:
        choice_data = user.recv(1024).decode('utf-8')
        if choice_data == "NEW":
            session_id = generate_id()
            while session_id in assigned_id:
                session_id = generate_id()
        else:
            parts = choice_data.split(":")
            if len(parts) > 1 and parts[1] != "":
                session_id = parts[1]
                found_in_logs = False
                if os.path.exists("chat_history.txt"):
                    with open("chat_history.txt", "r", encoding='utf-8') as file:
                        if session_id in file.read():
```

```
                            found_in_logs = True

                if found_in_logs and session_id not in assigned_id:
                    with open("chat_history.txt", "r", encoding='utf-8') as file:
                        for line in file:
                            user.send(("HIST:" + line.strip() + "\n").encode('utf-8'))
                            time.sleep(0.01)
                else:
                    session_id = generate_id()
                    while session_id in assigned_id:
                        session_id = generate_id()
            else:
                session_id = generate_id()
                while session_id in assigned_id:
                    session_id = generate_id()
    except Exception as e:
        print("ERROR przy logowaniu: " + str(e))
        user.close()
        return

    assigned_id.append(session_id)
    user.send(("SET_ID:" + session_id + "\n").encode('utf-8'))
    handle_broadcast(("Uzytkownik o ID " + session_id + " dolaczyl.").encode('utf-8'), user)
```

3.2.2. Plik client.py

Listing 3.3. Wątek odbiorczy z buforowaniem historii

```
def handle_messages():
    global my_id
    while True:
        try:
            received_bytes = client.recv(4096)
            if not received_bytes: break
            raw_data = received_bytes.decode('utf-8')
            messages = raw_data.split('\n')

            for msg in messages:
                if not msg: continue
                if msg.startswith("SET_ID:"):
                    my_id = msg.split(":")[1].strip()
                elif msg.startswith("HIST:"):
                    history_buffer.append(msg.replace("HIST:", "").strip())
                elif "dolaczyl" in msg or "wyszedl" in msg:
                    print("\r" + CLEAR + YELLOW + BOLD + msg + RESET)
                    print(GREEN + "TY (ID-" + my_id + "): " + RESET, end="", flush=True)
                else:
                    print("\r" + CLEAR + msg)
                    print(GREEN + "TY (ID-" + my_id + "): " + RESET, end="", flush=True)
        except Exception as e:
            print("ERROR: odebranie wiadomosci " + str(e))
            break
```

Listing 3.4. Synchronizacja i kolorowanie przywróconej historii

```
while my_id == "???":
    time.sleep(0.1)
```

```
decoration()
print(ASCII_LOGO)

if history_buffer:
    print(YELLOW + "--- PRZYWROCONA HISTORIA CZATU ---" + RESET)
    for h in history_buffer:
        if ("#" + my_id + ":") in h:
            print(GREEN + h + RESET)
        else:
            print(h)
    print(YELLOW + "----------------------------------" + RESET)
```

3.3. Moduły i oprogramowanie

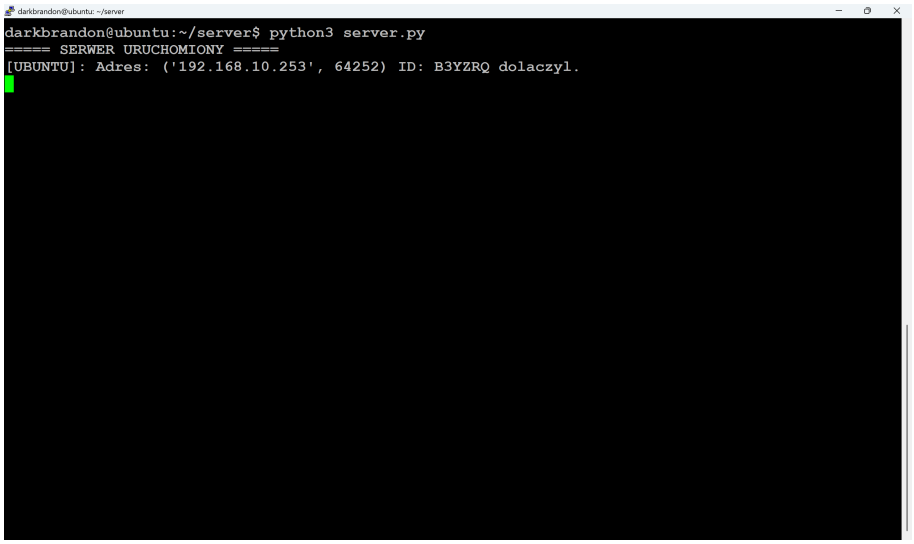
Moduły użyte w języku **Python** oraz całość użytego oprogramowania znajdują się na liście poniżej:

- Python 3.14.2
- Microsoft Windows 11
- Oracle VirtualBox 7.2.4
- Visual Studio Code
- Ubuntu Server 24.04.3 LTS
- Linux Mint 22.3 Cinnamon 64-bit
- Windows 11 LTSC
- PuTTY
- Moduły Python:
  - `socket`
  - `threading`
  - `random`
  - `string`
  - `os`
  - `time`

4. Prezentacja działania i testy

4.1. Serwer Ubuntu: komunikaty rozpoczęcia pracy i dołączenia

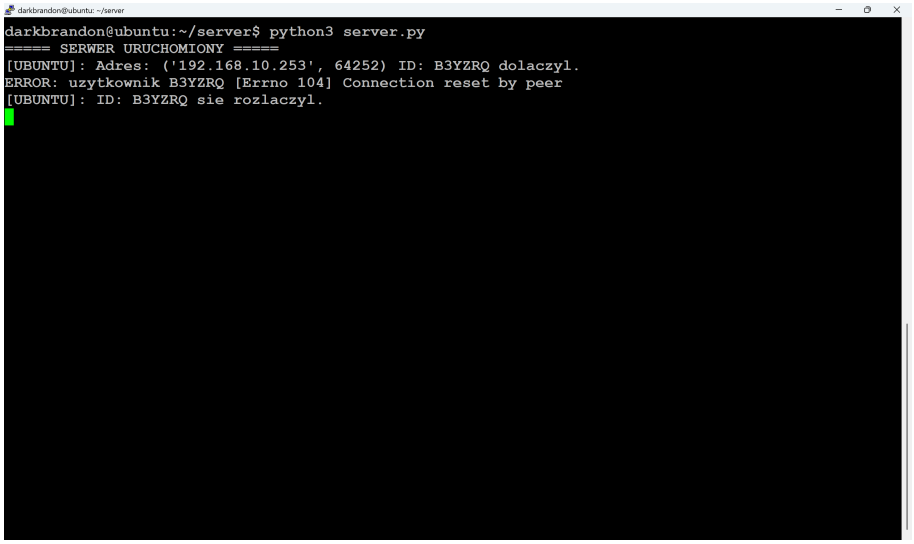
Działanie serwera wraz z komunikatami o dołączeniu się użytkowników są przedstawione na obrazku poniżej - **rys. 4.1**



Rys. 4.1. Działanie serwera i komunikat dołączenia użytkownika

4.2. Serwer Ubuntu: komunikaty rozłączania się użytkowników

Działanie serwera wraz z komunikatami o dołączeniu się użytkowników i ich następnym rozłączeniu są przedstawione na obrazku poniżej - **rys. 4.2**

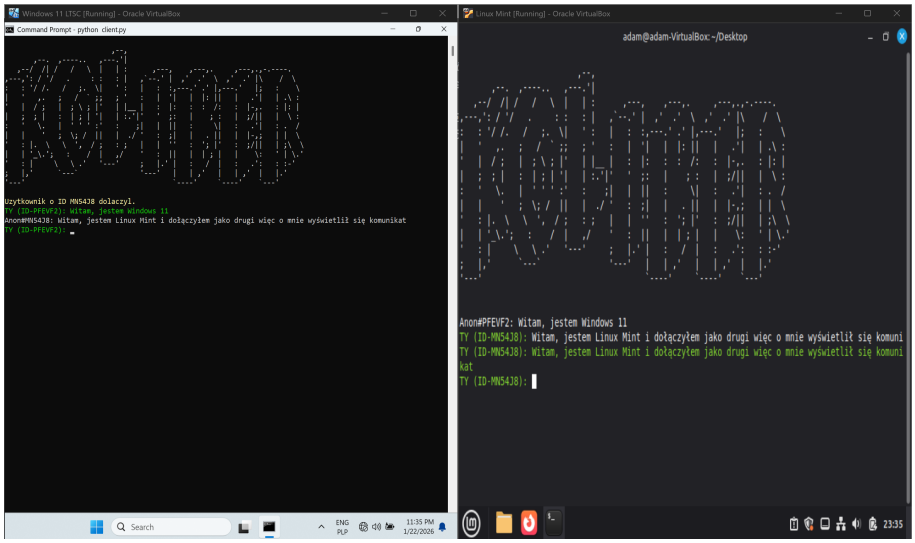


Rys. 4.2. Rozłączenie użytkownika - perspektywa serwera

4.3. Działanie aplikacji z perspektywy maszyn klienckich

4.3.1. Przesyłanie wiadomości i komunikat dołączenia

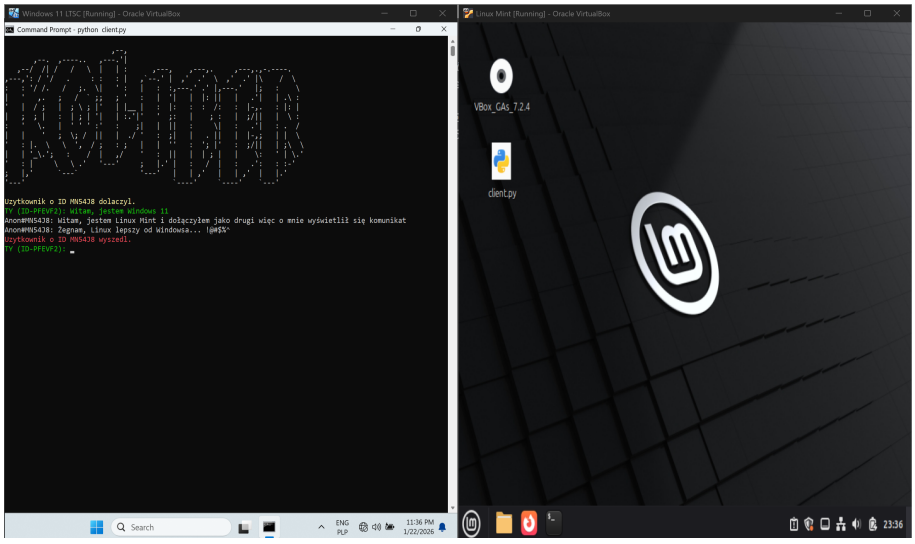
Działanie aplikacji po stronie użytkowników na maszynach `Windows 11 LTSC` i `Linux Mint`, gdzie widoczne jest przesyłanie wiadomości oraz komunikat o dołączeniu użytkownika widoczne jest na rysunku poniżej - **rys. 4.3**



Rys. 4.3. Przekazywanie wiadomości i dołączenie użytkownika

4.3.2. Rozłączenie się - komunikat

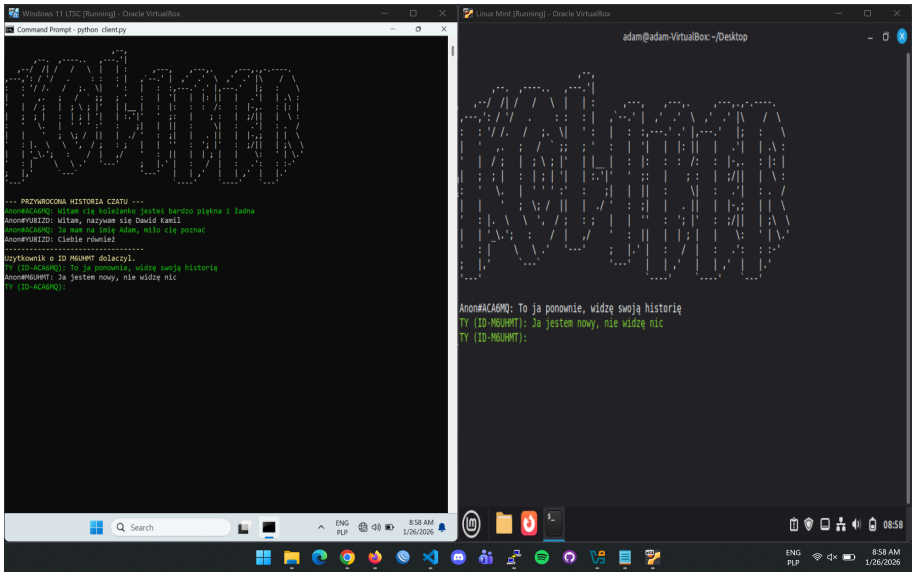
Komunikat informujący o rozłączeniu się jednego z użytkowników widoczny jest na obrazku poniżej - **rys. 4.4**



Rys. 4.4. Rozłączenie użytkownika - komunikat

4.3.3. Przywracanie historii - treść

Oboje użytkowników zaczęło nową sesję. Natomiast tylko użytkownik o ID `ACA6MQ` zapamiętał swoje ID i użyje je do przywrócenia konwersacji. Warto zobaczyć co dzieje się na terminalu maszyny `Linux Mint`, która rozpoczęła nową sesje. Wynik operacji widoczny jest na obrazku poniżej - **rys. 4.5**



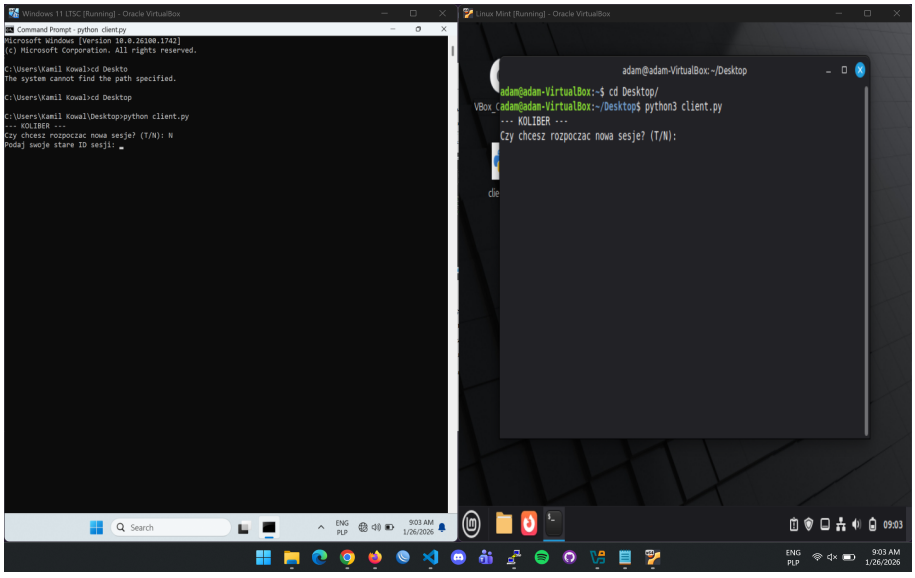
Rys. 4.5. Przywrócenie historii, gdzie konkretny ID uczestniczył

4.3.4. Dostępność historii dla nowych użytkowników - zachowanie

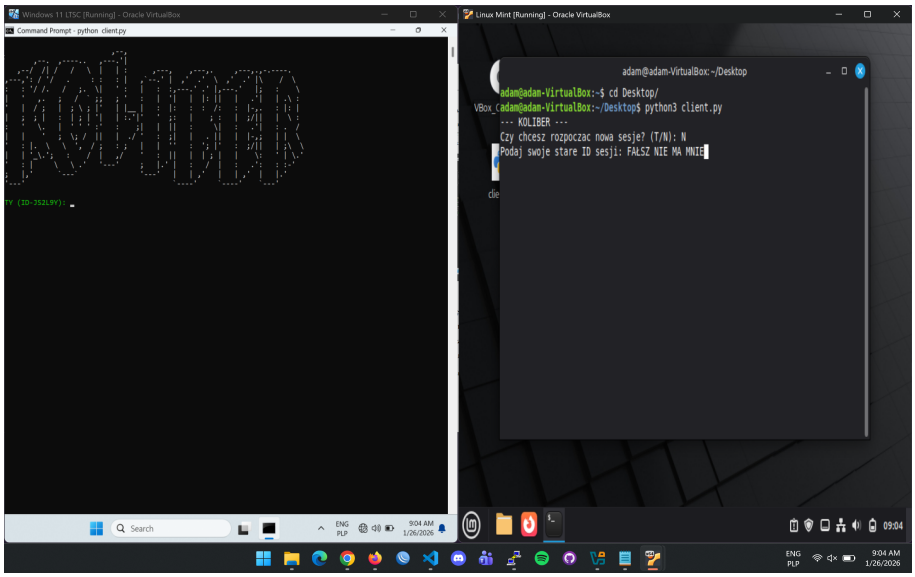
Zachowanie to przedstawione jest na obrazku powyżej - **rys. 4.5**

4.3.5. Zabezpieczenie - co jeśli ktoś jest nowy i chce przywrócić sesję?

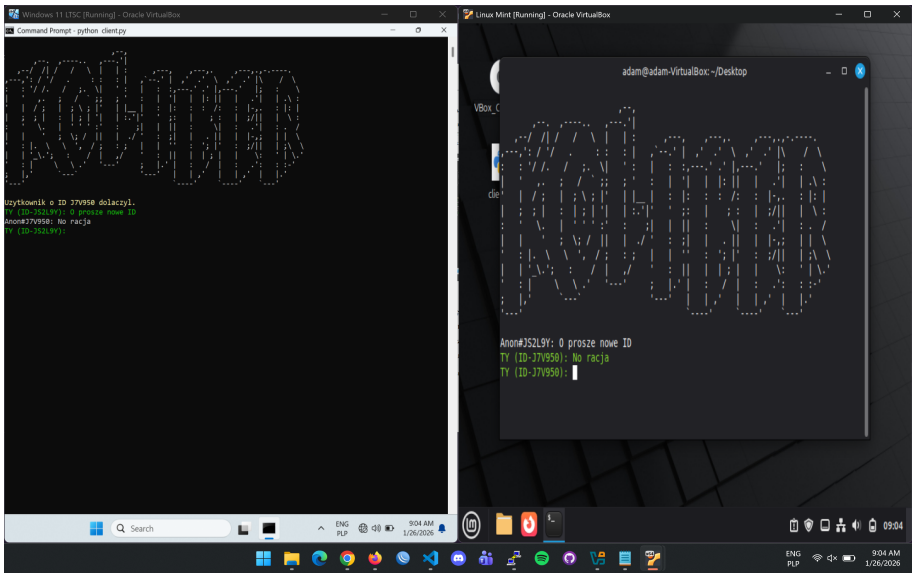
W przypadku, gdy ktoś uzna, że nie chce zacząć nowej sesji i wpisze nieistniejące ID, zostanie mu przydzielone nowe ID losowe co spowoduje ten sam proces co przy stworzeniu nowej sesji (Opcja TAK). Efekt widoczny jest na obrazkach poniżej - **rys. 4.6, rys. 4.7, rys. 4.8**



Rys. 4.6. Czy chcę rozpocząć nową sesję? (N)



Rys. 4.7. Podanie nieistniejącego ID



Rys. 4.8. Efekt końcowy - rozpoczęcie nowej sesji z losowymi ID

5. Instrukcja uruchomieniowa

Aby poprawnie uruchomić skrypty a co za tym idzie kompletną aplikację, należy przeprowadzić następujące kroki:

1. Zainstalować środowisko **VirtualBox**, najlepiej w najnowszej wersji.
2. Utworzyć maszynę wirtualną - Ubuntu Server. W tym przypadku wersja `Ubuntu 24.04.3 LTS`.
3. Utworzyć klienckie maszyny - w przypadku projektu Windows 11 LTSC oraz Linux Mint.
4. Plik `client.py` przesłać do maszyn klienckich np. na pulpit.
5. Plik `server.py` przesłać na serwer Ubuntu.
6. Plik `server.py` uruchomić na serwerze poprzez komendę `python3 server.py` - po wykonaniu komendy powinien przywitać nas odpowiedni komunikat.
7. Plik `client.py` uruchomić poprzez komendę `python client.py (Windows) / python3 client.py (Linux)` uprzednio przechodząc w odpowiednią ścieżkę w terminalu - CMD / Bash.
8. Po wykonaniu wcześniejszych kroków całość powinna działać poprawnie.

6. Podsumowanie

Zrealizowany projekt pozwolił na stworzenie w pełni funkcjonalnego, wielowątkowego komunikatora sieciowego. Wszystkie założenia oraz cele zostały w pełni osiągnięte. Podczas prac szczególną uwagę poświęcono stabilności połączenia oraz czytelności "GUI"w terminalu poprzez kolory odróżniające komunikaty. W przyszłości można dodać bardziej rozbudowany interfejs użytkownika poprzez odpowiednie moduły oraz opcję przesyłania plików między użytkownikami.

Bibliografia

[1] NeuralNine, 2026. *https://www.youtube.com/@NeuralNine*.

[2] Python, 2026. *https://docs.python.org/3/library/socket.html*.

[3] Stack Overflow, 2026. *https://stackoverflow.com/*.

[4] W3Schools, 2026. *https://www.w3schools.com/*.

Spis rysunków