

Politechnika Wrocławska  
Wydział Elektroniki  
Organizacja i Architektura Komputerów

---

# SERWER INTERNETOWY

---

*Autorzy:*  
KONRAD PIECHOTA  
235968

Prowadzący projekt  
mgr Daniel Cieszko

16.06.2019

# Spis treści

<b>1</b>	<b>Wstęp</b>	<b>2</b>
1.1	Wstęp teoretyczny . . . . .	2
1.2	Założenia projektu . . . . .	2
<b>2</b>	<b>Realizacja projektu</b>	<b>2</b>
2.1	Moduł klienta . . . . .	2
2.2	Moduł serwera . . . . .	4
2.3	Moduł serwisowy . . . . .	5
<b>3</b>	<b>Testy</b>	<b>6</b>
3.1	Sposób testowania . . . . .	6
3.2	Przebieg . . . . .	6
<b>4</b>	<b>Wnioski</b>	<b>7</b>
4.1	Podsumowanie . . . . .	7
4.2	Wnioski . . . . .	7

# 1 Wstęp

## 1.1 Wstęp teoretyczny

FTP z angielskiego File Transfer Protocol jest to protokół komunikacyjny pozwalający na przesył danych między użytkownikiem a serwerem. Aby mogło nastąpić połączenie użytkownik musi znać adres serwera, oraz posiadać dane logowania chyba, że serwer udostępnia opcję połączenia anonimowego. Użytkownik inicjuje połączenie z serwerem za pomocą którego wysyłane są polecenia. W połączeniu serwer korzysta z zarezerwowanego portu 21, natomiast użytkownik z losowo przydzielonym, zwykle większym od 1024.

Raspberry Pi jest to minikomputer wyposażony w procesor Quad Core 1.2GHz BARM-8 Cortex-A53 64bit, 1Gb pamięci RAM ponadto posiada wbudowany moduł WiFi i Bluetooth.

## 1.2 Założenia projektu

Projekt zakładał zrealizowanie serwera internetowego bazującego na protokole komunikacyjnym FTP w języku Python 3, a następnie jego implementacja na urządzenie Raspberry Pi 3B oraz zbadanie jej działania w zależności od ilości klientów.

# 2 Realizacja projektu

Aplikacja została podzielona na 3 następujące moduły:

- Moduł klienta
- Moduł serwera
- Moduł serwisowy

## 2.1 Moduł klienta

W aplikacji, moduł klienta pełni rolę interfejsu użytkownika - odpowiedzialny jest za połączenie klienta z serwerem, a także dalszą obsługę tj. wysyłanie danych logowania [Listing 1], odbierania plików [Listing 2], wysyłania plików. Jest to jedyny moduł znajdujący się po stronie użytkownika. Poniżej znajduje się grafika przedstawiająca logi klienta który połączył się do serwera, a następnie zażądał pobrania pliku 'file.txt' [Rysunek 1]

```
You must log in.  
Host:  
192.168.1.7  
Login:  
test  
Password:  
test  
b'Authentication passed'  
b'What do you want to do? \n Type "send" to send file. Type "get" to download file.'  
get  
['Available files: file.txt 2.txt']  
file.txt  
Downloaded succesfully
```

Rysunek 1: Przykładowe logi klienta

Listing 1: Funkcja realizująca logowanie po stronie klienta

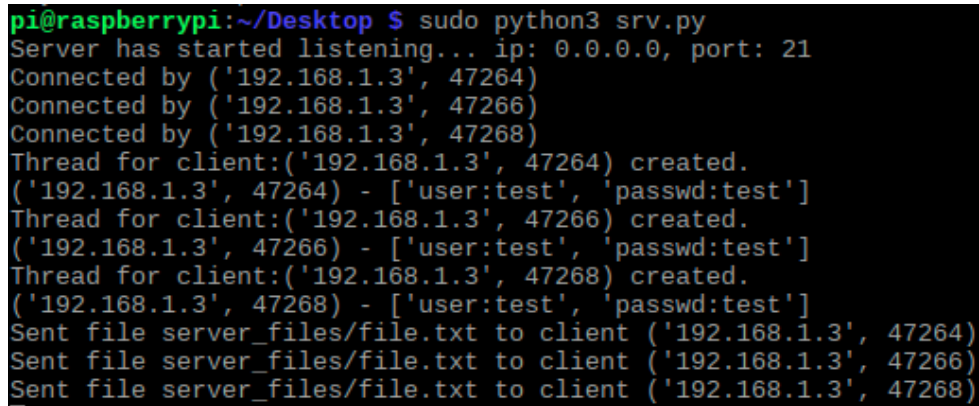
```
def login(self):  
    print("Login:")  
    login = input()  
    print("Password:")  
    passwd = input()  
    packet = "user:{},passwd:{}".format(login, passwd)  
    self.s.sendall(packet.encode("utf-8"))  
    data = self.s.recv(1024)  
    print(data)
```

Listing 2: Funkcja realizująca pobieranie pliku z serwera

```
def download_file(self):  
    recv_data = self.s.recv(1024)  
    files = recv_data.decode('utf-8').strip().split(",")  
    print(files)  
    file_name = self.file  
    self.s.sendall(file_name.encode('utf-8'))  
    recv_data = self.s.recv(1024)  
    file_size = recv_data.decode('utf-8').strip().split(",")  
    save_file = open("client_files/{}.txt"\br/>        .format(self.id), "w+")  
    amount_recieved_data = 0  
    self.s.sendall(("Confirmed").encode('utf-8'))  
    while amount_recieved_data < float(file_size[1]):  
        recv_data = self.s.recv(8)  
        amount_recieved_data += len(recv_data)  
        save_file.write(recv_data.decode('utf-8'))  
    message = 'Downloaded_succesfully '  
    print(message)  
    self.s.sendall(message.encode('utf-8'))  
    save_file.close()
```

## 2.2 Moduł serwera

Moduł serwera jest głównym elementem aplikacji, jego zadaniem jest nasłuchiwanie czy pojawiają się zgłoszenia od klientów chętnych połączyć się z serwerem, w przypadku połączenia się z klientem serwer tworzy proces modułu serwisowego. Poniżej znajduje się przykład logów pojawiających się na działającym serwerze na urządzeniu Raspberry Pi [Rysunek 2]



```
pi@raspberrypi:~/Desktop $ sudo python3 srv.py
Server has started listening... ip: 0.0.0.0, port: 21
Connected by ('192.168.1.3', 47264)
Connected by ('192.168.1.3', 47266)
Connected by ('192.168.1.3', 47268)
Thread for client:('192.168.1.3', 47264) created.
('192.168.1.3', 47264) - ['user:test', 'passwd:test']
Thread for client:('192.168.1.3', 47266) created.
('192.168.1.3', 47266) - ['user:test', 'passwd:test']
Thread for client:('192.168.1.3', 47268) created.
('192.168.1.3', 47268) - ['user:test', 'passwd:test']
Sent file server_files/file.txt to client ('192.168.1.3', 47264)
Sent file server_files/file.txt to client ('192.168.1.3', 47266)
Sent file server_files/file.txt to client ('192.168.1.3', 47268)
```

Rysunek 2: Przykładowe logi serwera

Listing 3: Moduł serwera

```
def run():
    while True:
        client, addr = s.accept()
        print('Connected by', addr)
        x = Process(target=Service, args=(client, addr))
        x.start()

HOST = socket.gethostname()
PORT = 21
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = (HOST, PORT)
s.bind(('', PORT))
s.listen()
ip, port = s.getsockname()
print("Server has started listening ... ip:\n{}\nport: {}".format(ip, PORT))
while True:
    client, addr = s.accept()
    print('Connected by', addr)
    x = Process(target=Service, args=(client, addr))
    x.start()
```

## 2.3 Moduł serwisowy

Po utworzeniu modułu przez serwer zajmuje się on obsługą klienta. Jego pierwszym zadaniem jest autoryzacja dostępu - sprawdzenie czy login i hasło podane przez klienta są poprawne [Listing 4], jeśli tak klient może przejść do pobierania lub wysyłania plików [Listing 5].

Listing 4: Autoryzacja logowania

```
def login(self):
    data = self.client.recv(1024)
    login_info = data.decode('utf-8')\
        .strip().split(",")

    print('{}-{}'.format(self.addr, login_info))
    user_info = login_info[0].split(":")
    pass_info = login_info[1].split(":")

    if user_info[1] in self.users:
        if (self.users[user_info[1]] == pass_info[1]) == True:
            message = "Authentication_passed"
            self.client.sendall(message.encode('utf-8'))
        else:
            message = "Authentication_failed.-Disconnecting"
            self.client.sendall(message.encode('utf-8'))
            sys.exit(1)
```

Listing 5: Wysyłanie pliku z serwera do klienta

```
def send_file(self):
    [...]
    self.client.sendall(package)
    recv_data = self.client.recv(1024)
    file = recv_data.decode('utf-8').strip().split(",")
    file_location = "server_files/" + file[0]
    file_size = os.path.getsize(file_location)
    self.client.sendall("Exists,{}\n"
        .format(file_size).encode('utf-8'))
    recv_data = self.client.recv(1024)
    with open(file_location, "rb") as file:
        self.client.sendfile(file)
    recv_data = self.client.recv(1024)
    print('Sent_file-{}-to-client-{}\n'
        .format(file_location, self.addr))
```

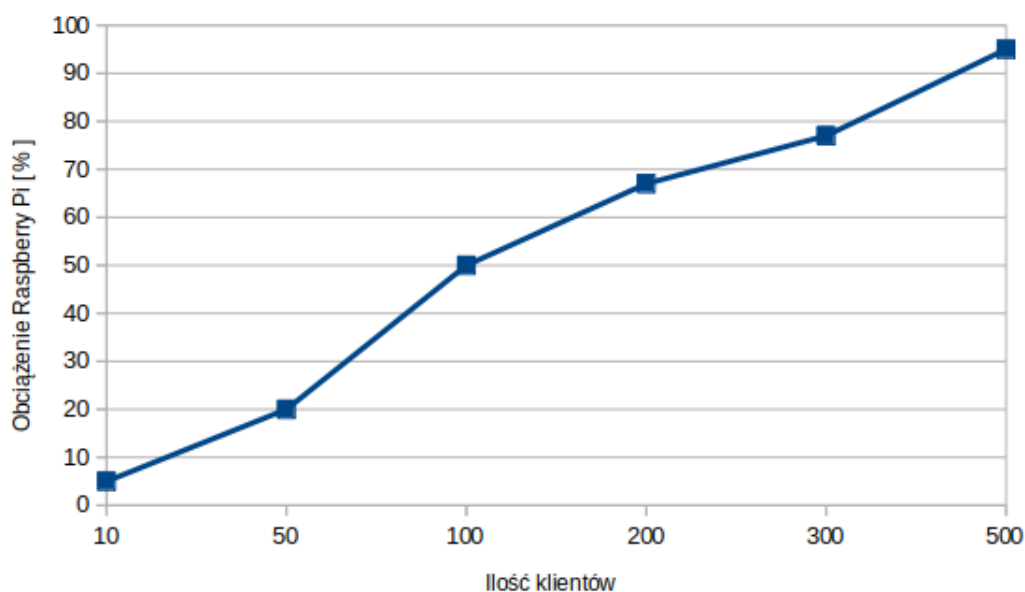
## 3 Testy

### 3.1 Sposób testowania

Testy polegały na jednoczesnym połączeniu n-klientów do serwera oraz zażądaniu przez nich pobrania tego samego pliku.

### 3.2 Przebieg

Wykonano 10 pomiarów dla 10, 50, 100, 200, 300 i 500 klientów, uzyskane wyniki zostały uśrednione. Na poniższym wykresie można zaobserwować zależność zużycia CPU Raspberry Pi w zależności od ilości klientów dla pliku tekstowego o rozmiarze 50kB



Rysunek 3: Wykres wydajności Raspberry Pi od ilości klientów

Zużycie procesora komputera Raspberry Pi wykazuje zależność zbliżoną do liniowej.

Dla większej ilości klientów wyniki są niemiernie z powodu występującego dla wielu klientów błędu [Listing 6]. Urządzenie Raspberry Pi pod wpływem dużego obciążenia nie rozpoznaje połączeń, a następnie je zamyka.

Listing 6: Błąd połączenia

```
ConnectionResetError: [Errno 104] Connection reset by peer
```

## **4 Wnioski**

### **4.1 Podsumowanie**

Zaimplementowana aplikacja serwer-klient działała poprawnie i wykazała się stabilnością względem rosnącej ilości klientów do poziomu około 500 klientów. Było to możliwe dzięki wykorzystaniu w module serwera multiprocessingu. Multithreading pozwolił na jednoczesne połączenie do około 280 klientów.

### **4.2 Wnioski**

Protokół FTP we współczesnej sieci internetowej jest podatny na wiele ataków m.in. sniffing lub DoS. Aby im zapobiec można skorzystać z szyfrowanej usługi FTP - FTPS, oraz ograniczyć dostęp do serwera dla klientów z jednego adresu IP.

Aplikację można uzupełnić o możliwość przesyłu plików innych od tekstowych np. obrazy.