LAPORAN PRAKTIKUM

PEMROGRAMAN JARINGAN

Tugas dan Latihan

Pengenalan Socket pada Python

Disusun oleh:

Pandu Rafa Panatagama (1203220063)

IF 02-01

PRODI INFORMATIKA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY SURABAYA

MARET 2024

Tugas dan Latihan

Analisa Hasil Percobaan tadi yang sudah dibuat Komunikasi antar socket dan Socket Option

1. Bagaimana cara membuat objek socket pada Python?

```
# Mengimpor modul socket
import socket

# Menginisialisasi socket untuk suatu entitas (server atau client)
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Modul socket menyediakan cara bekerja dengan socket. Socket memungkinkan pembuatan koneksi jaringan melalui berbagai protokol jaringan seperti TCP atau UDP untuk mengirim dan menerima data. Pada contoh di atas, objek socket dibuat dengan fungsi socket(socket_family, socket_type), fungsi ini mengembalikan objek soket yang metodenya mengimplementasikan berbagai panggilan sistem soket.

2. Apa yang dimaksud dengan alamat pada socket?

```
server_address = ('localhost', 8000)
```

Alamat soket adalah pengidentifikasi unik yang terdiri dari dua bagian: alamat IP dan nomor port. Kombinasi kedua elemen ini memungkinkan terjadinya komunikasi jaringan antar perangkat. Pada contoh di atas, alamat IP yang digunakan adalah 127.0.01 atau `localhost`. Alamat IP ini menunjuk ke server yang berjalan pada perangkat yang sama.

3. Bagaimana cara mengikat objek socket ke alamat tertentu pada Python?

```
server_socket.bind(('localhost', 8000))
```

Fungsi bind(address, port) digunakan untuk mengikat socket ke alamat dan nomor port tertentu di host. Pada contoh di atas, server_socket.bind(`localhost`, 8000) digunakan untuk mengikat server_socket ke alamat 'localhost' dan port 8000. Setelah diikat, socket siap mendengarkan koneksi yang masuk pada alamat dan port tersebut.

4. Bagaimana cara menerima koneksi dari klien pada server menggunakan Python?

```
server_socket.listen(5)

print("Waiting for connection...")
client_socket, client_address = server_socket.accept()
print(f"Connection from {client_address}")
```

Fungsi listen(backlog) pada dasarnya menetapkan tanda pada struktur socket internal yang menandai socket tersebut sebagai socket pendengar pasif, yang dapat dilanjutkan ke pemanggilan fungsi accept(). Ini membuka port yang telah terikat sehingga socket dapat mulai menerima koneksi dari client.

Argumen backlog menentukan jumlah maksimum koneksi yang diantri dan minimal harus 1, nilai maksimum bergantung pada sistem (umumnya 5).

Fungsi accept() meminta soket pendengar untuk menerima koneksi yang masuk berikutnya dan mengembalikan deskriptor socket untuk koneksi tersebut.

5. Bagaimana cara mengirim dan menerima pesan menggunakan socket pada Python?

```
# mengirim data ke server
message = 'Hello, server!'
client_socket.sendall(message.encode())

# menerima respons dari server
data = client_socket.recv(1024)
print(f'Received: {data.decode()}')
```

Pesan dikirim dengan fungsi sendall() dan menerima pesan dengan fungsi recv(). Fungsi sendall() digunakan untuk mengirimkan seluruh byte pesan. Karena itu, pesan yang akan dikirim (string) perlu diubah menjadi byte menggunakn fungsi encode(). Fungsi sendall() akan mencoba mengirim seluruh data, mengulangi jika diperlukan, hingga semua data terkirim.

Fungsi recv(1024) mengambil data yang dikirimkan oleh server melalui socket dengan ukuran buffer 1024 byte. Respons dari server kemudian disimpan dalam variabel data, yang nantinya akan diterjemahkan dengan fungsi decode() sebelum ditampilkan.

6. Apa yang dimaksud dengan socket option pada Python, dan berikan contohnya?

Socket option adalah parameter yang dapat diatur untuk mengkonfigurasi perilaku socket. Socket options memungkinkan pengguna untuk mengendalikan berbagai aspek koneksi socket, dan dapat digunakan untuk mengoptimalkan atau menyesuaikan perilaku socket sesuai dengan kebutuhan aplikasi atau skenario jaringan tertentu.

```
import socket

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

└> SO_REUSEADDR: Memungkinkan penggunaan kembali alamat lokal yang sama oleh socket yang baru setelah socket yang lama ditutup.

```
import socket

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.setsockopt(socket.SOL_SOCKET, socket.SO_KEEPALIVE, 1)
```

└> SO_KEEPALIVE: Mengaktifkan atau menonaktifkan mekanisme "keep-alive" untuk koneksi TCP, yang secara otomatis mengirim paket kecil untuk memeriksa apakah koneksi masih hidup.

```
import socket

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.settimeout(10)  # Timeout dalam 10 detik
```

└> SO_TIMEOUT: Menetapkan batas waktu (timeout) untuk operasi tertentu di socket, seperti recv().

```
import socket

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)
```

└> TCP_NODELAY: Mengaktifkan atau menonaktifkan algoritma Nagle untuk mengontrol penundaan pengiriman dalam koneksi TCP.

```
import socket

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Mengatur opsi SO_LINGER
linger_option = socket.LingerOption(True, 30)  # True: menunggu, 30: waktu
penundaan dalam detik
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_LINGER, linger_option)
```

└> SO_LINGER: Mengatur apakah socket harus menunggu untuk mengirimkan data yang tersisa sebelum ditutup. Opsi ini biasanya digunakan dalam situasi di mana aplikasi ingin memastikan bahwa semua data yang belum terkirim dikirimkan sebelum socket ditutup.

7. Bagaimana cara mengirim pesan pada socket menggunakan protokol TCP pada Python?

```
import socket

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(('localhost', 12345))
```

Pertama, socket yang dibuat (bertipe SOCK_STREAM untuk TCP) perlu dikoneksikan dengan alamat dan port tujuan menggunakan metode connect(). Hal ini merupakan langkah awal three-way handshake yang diperlukan TCP sebagai protocol berorientasi koneksi.

```
client_socket.send("Hello, Server!".encode('utf-8'))
client_socket.close()
```

Pesan yang akan dikirim perlu disandikan terlebih dahulu, lalu pesan siap dikirim. Selanjutnya, metode close() akan menutup koneksi Socket dan melepaskan semua sumber daya terkait dengan batas waktu yang ditentukan untuk memungkinkan pengiriman data antrean.

8. Bagaimana cara menerima pesan pada socket menggunakan protokol TCP pada Python?

```
import socket

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(('localhost', 12345))
```

Pertama, socket yang telah dibuat (bertipe SOCK_STREAM untuk TCP) diikat pada alamat dan port tertentu di host.

```
server_socket.listen(5)
```

Berikutnya, socket disiapkan sebagai pendengar pasif dan dilanjutkan ke fase membuka port dengan menjalankan metode accept().

```
while True:
    client_socket, address = server_socket.accept()
    data = client_socket.recv(1024)
    print(f"Received data: {data.decode('utf-8')}")
    client_socket.close()
```

Dengan demikian, socket siap menyambut koneksi dari client (proses handshake). Untuk benar-benar menerima pesan dari client (pengirim pesan), dijalanakn metode recv() yang mampu membaca pesan yang dikirimkan melalui socket.

9. Bagaimana cara mengirim pesan pada socket menggunakan protokol UDP pada Python?

```
import socket

# Membuat socket UDP
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

Pertama, socket dibuat dengan tipe SOCK_DGRAM untuk membuat socket UDP. UDP adalah protocol tanpa koneksi. Setiap pesan dikirimkan secara independen, tanpa membangun koneksi terlebih dahulu.

```
server_address = ('localhost', 12345)
```

Selanjutnya, alamat dan port yang akan dituju disiapkan.

```
while True:
    message = input("Enter message: ")
    client_socket.sendto(message.encode('utf-8'), server_address)
```

Akhirnya, pesan yang telah disandikan, dikirim ke alamat yang telah disiapkan.

10. Bagaimana cara menerima pesan pada socket menggunakan protokol UDP pada Python?

```
import socket

# Membuat socket UDP
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

Pertama, socket dibuat dengan tipe SOCK_DGRAM untuk membuat socket UDP.

```
server_address = ('localhost', 12345)
server_socket.bind(server_address)
print("Server is ready to receive messages.")
```

Selanjutnya, socket diikat ke alamat dan port yang telah disiapkan.

```
while True:
    data, client_address = server_socket.recvfrom(1024)
    print(f"Received message from {client_address}: {data.decode('utf-8')}")
```

Akhirnya, socket bisa langsung menerima pesan yang dikirm client (pengirim pesan) menggunakan metode recvfrom(). Metode ini dapat membaca pesan atau lebih tepatnya sejumlah byte (dalam contoh ini 1024 byte) yang dikirim dari soket UDP.