CW8691 Practicals Experiments Code In Python with algorithms

May 12, 2023

0.1 HTTP WEB CLIENT PROGRAM TO DOWNLOAD A WEB PAGE USING TCP SOCKETS (Exp No 2)

1. Import the socket module to create a socket object and specify the IP address and port number of the remote host to connect to.
2. Connect to the remote host using the connect() method of the socket object and send a GET request to retrieve the web page from the specified host.
3. Use the sendall() method to send the GET request to the server.
4. Use the recv() method of the socket object to receive the response from the server and decode it into a string using the UTF-8 encoding.
5. Print the received response to the console.

[ ]:

```
import socket
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(("aurcc.ac.in",80))
s.sendall(b"GET /HTTP/1.1\r\nHost:www.aurcc.ac.in\r\nAccept:text/html\r\n\r\n")␣
  ↪print(str(s.recv(4096),'utf-8'))
```

0.2 Program for Server and Client (Exp No 3a)

Here’s a simple algorithm for a client-server communication using TCP sockets:

Server:

1. Create a socket object using socket.socket()
2. Bind the socket to a specific address and port using socket.bind()
3. Listen for incoming connections using socket.listen()
4. Accept a connection from a client using socket.accept()
5. Receive data from the client using socket.recv()
6. Process the data as needed
7. Send a response to the client using socket.send()
8. Close the connection using socket.close()

Client:

1. Create a socket object using socket.socket()
2. Connect to the server using socket.connect()
3. Send data to the server using socket.send()
4. Receive a response from the server using socket.recv()
5. Process the response as needed
6. Close the connection using socket.close()

0.2.1 Server

[ ]:

```
import socket
HOST = ''  # Empty string means any available interface
PORT = 65432  # Port to listen on
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    while True:
        conn, addr = s.accept()
        with conn:
            print('Connected by', addr)
            data = conn.recv(1024)
            conn.sendall(data)
```

0.2.2 Client

[ ]:

```
import socket
HOST = 'localhost'  # Server IP address or hostname
PORT = 65432  # Server port
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b'Hello, world')
    data = s.recv(1024)
print('Received', repr(data))
```

0.3 Program For Server-client Chatbot(Exp No 3b)

Here is a simple algorithm for a server-client chatbot using TCP sockets:

Server Side:

1. Create a socket object using socket() function and bind it to a particular IP address and port number.
2. Listen for incoming connections using the listen() function.
3. Accept incoming connections from clients using the accept() function, which will return a new socket object for each client.
4. Create a loop to handle the incoming client connections.

5. Receive messages from clients using the recv() function.
6. Process the message and generate a response.
7. Send the response back to the client using the send() function.
8. Close the connection to the client using the close() function.

Client Side:

1. Create a socket object using socket() function and connect it to the server IP address and port number.
2. Create a loop to allow for multiple messages to be sent to the server.
3. Prompt the user to enter a message and send it to the server using the send() function.
4. Receive the response from the server using the recv() function.
5. Display the response to the user.
6. Close the connection to the server using the close() function.

0.3.1 Server

[ ]:

```python
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = ('localhost', 8888)
print('Starting up on {} port {}'.format(*server_address))
sock.bind(server_address)
sock.listen(1)
print('Waiting for a connection...')
connection, client_address = sock.accept()
print('Connection from', client_address)
while True:
    data = connection.recv(1024)
    print('Client: {}'.format(data.decode()))
    message = input('Server: ')
    connection.sendall(message.encode())
```

0.3.2 Client

[ ]:

```python
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = ('localhost', 8888)
print('Connecting to {} port {}'.format(*server_address))
sock.connect(server_address)
while True:
    message = input('Client: ')
    sock.sendall(message.encode())
    data = sock.recv(1024)
    print('Server: {}'.format(data.decode()))
```

0.4 Program for DNS (Exp No 4)

Here is a simple algorithm for the Simulation of DNS using UDP sockets:

1. Create a socket using `socket.socket()` and pass `socket.AF_INET` and `socket.SOCK_DGRAM` as parameters to create a UDP socket.
2. Bind the socket to a specific IP address and port using `socket.bind()`.
3. Define a dictionary containing the mapping between domain names and IP addresses.
4. Start an infinite loop to listen for incoming DNS requests. Receive the request data and client address using `socket.recvfrom()`.
5. Decode the received data to get the domain name.
6. If the domain name is present in the dictionary, retrieve the corresponding IP address and send it back to the client using `socket.sendto()`.
7. If the domain name is not present in the dictionary, send an error message back to the client using `socket.sendto()`.
8. Print the received request and sent response messages for debugging purposes.
9. Close the socket using `socket.close()`.

[ ]:

```
import socket
dns_ip = '127.0.0.1'
dns_port = 5353
dns_table = {
    'google.com': '216.58.194.174',
    'amazon.in': '52.95.116.115'
}
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((dns_ip, dns_port))
print('DNS server listening on {}:{}'.format(dns_ip, dns_port))
while True:
    data, addr = sock.recvfrom(1024)
    domain = data.decode('utf-8').strip()
    if domain in dns_table:
        ip = dns_table[domain]
        sock.sendto(ip.encode('utf-8'), addr)
        print('Sent DNS response for {}: {}'.format(domain, ip))
    else:
        error = 'Domain not found'
        sock.sendto(error.encode('utf-8'), addr)
        print('Sent DNS error response for {}: {}'.format(domain, error))
```

0.5 ARP and RARP (Exp No 5)

Sure, here's a simple algorithm for ARP and RARP:

Algorithm for ARP:

1. The sender wants to send a packet to a destination machine whose MAC address is unknown to the sender.
2. The sender first checks if the destination IP address is on the local network. If not, it sends the packet to the default gateway.
3. The sender sends an ARP request broadcast message to the local network asking for the MAC address corresponding to the destination IP address.
4. The destination machine with the corresponding IP address responds to the ARP request with its MAC address.
5. The sender receives the MAC address of the destination machine from the ARP response message.
6. The sender caches the MAC address in its ARP table to use it for future communication with the destination machine.

Algorithm for RARP:

1. The sender wants to obtain its IP address from the RARP server.
2. The sender sends a RARP request broadcast message on the local network.
3. The RARP server responds to the RARP request with the IP address corresponding to the sender's MAC address.
4. The sender receives its IP address from the RARP response message.
5. The sender caches the IP address in its RARP table to use it for future communication.

0.5.1 ARP

[ ]:

```
from scapy.all import *

# Send an ARP request to get the MAC address of a given IP address
def arp_request(ip):
    arp = ARP(pdst=ip)
    ether = Ether(dst='ff:ff:ff:ff:ff:ff')
    packet = ether/arp
    result = srp(packet, timeout=3, verbose=0)[0]
    for sent, received in result:
        return received.hwsrc

# Test the ARP function
ip = '192.168.0.1' # Replace with your target IP address
mac = arp_request(ip)
print('The MAC address of', ip, 'is', mac)
```

0.5.2 RARP

[ ]:

```
from scapy.all import *

# Send a RARP request to get the IP address of a given MAC address
def rarp_request(mac):
    arp = ARP(op=2, hwsrc=mac)
    ether = Ether(dst='ff:ff:ff:ff:ff:ff', type=0x806)
    packet = ether/arp
    result = srp(packet, timeout=3, verbose=0)[0]
    for sent, received in result:
        return received.psrc

# Test the RARP function
mac = '00:11:22:33:44:55' # Replace with your target MAC address
ip = rarp_request(mac)
print('The IP address of', mac, 'is', ip)
```