Experiment No. 9

Date - 12/06/2021

Concurrent Time Server Application using UDP

Aim:

To write a program to Implement Concurrent Time Server application using UDP to execute the program at a remote server. Client sends a time request to the server, server sends its system time back to the client and is displayed by the client.

THEORY:

A concurrent server handles multiple clients at the same time. The simplest technique for a concurrent server is to call the fork function, creating one child process for each client. An alternative technique is to use threads instead.

1. When a connection request arrives in the main process of a concurrent server, it schedules a child process and forwards the connection to the child process.

2. The child process takes the connection from the main process.

3. The child process receives the client request, processes it, and returns a reply to the client.

4. The connection is closed, and the child process terminates or signals to the main process that it is available for a new connection.

Algorithm

a) Server

1. Start
2. Import modules os, socket
3. Initialize the host and port
4. Create socket sock using socket function
5. Bind the socket with host and port
6. Server initiates listening phase
7. If time requests from clients are received the server creates a child to handle the request.
8. Server child accepts the request and sends the system time to the requested client.
9. Stop

b) Client

1. Start
2. Import module socket
3. Initialize host and port
4. Create the socket sock using socket function
5. Socket is connected to the host and port
6. Client send the time request to server
7. Client receives the system time from client and prints the time
8. Stop

Program

a) Server

```
import os, socket

from signal import signal, SIGPIPE, SIG_DFL

from datetime import datetime

host="127.0.0.1"

port=8089

sock=socket.socket(family=socket.AF_INET, type=socket.SOCK_STREAM)

sock.bind((host, port))

sock.listen(10)

def handle_client(sock, addr, i):

        while True:

                data =sock.recvfrom(1024)

                if not data:

                        print("Connection with client " + str(i) + " has been broken\n")

                        break

                now = str(datetime.now())

                sock.sendto(now,(host, port))

                signal(SIGPIPE, SIG_DFL)

def server():

        i=1

         while i<=10:

                 c, addr=sock.accept()
                 child_pid=os.fork()
                 if child_pid==0:
                     print("Connection is successful with client " + str(i)+str(addr)+ "\n")
                     handle_client(c, addr, i)
                     break
                 else:
                        i+=1
server()
```

b) Client

```
import socket
def client():
    host="127.0.0.1"
    port=8089
    sock=socket.socket(family=socket.AF_INET, type=socket.SOCK_STREAM)
    sock.connect((host, port))
    sock.sendto("Send time",(host, port))
    print("The system time is :\n")
    res= sock.recvfrom(1024)
    print(res)
client()
```

Output

a) Server

```
$python E9s.py

Connection Successful with client  1 ('127.0.0.1', 59114)

Connection Successful with client  2 ('127.0.0.1', 59116)
```

b) Client1

```
$python E9c.py

The system time is :

('2021-06-11 18:45:57.430034', None)
```

c) Client2

```
$python E9c.py

The system time is :

('2021-06-11 18:45:57.430034', None)
```

Result

Successfully executed the program to Implement Concurrent Time Server application using UDP to execute the program at a remote server.