Electric Bill Processing

With Client-Server Architecture

Operating System (CS450) Event Project Report

Kartik N Nayak.

4th sem., 'C' sec.

Roll no.: 17, USN: 01JST18CS047

CS&E, JSSSTU, Mysore

Keerthana P B.

4th sem., 'C' sec.

Roll no.: 18, USN: 01JST18CS048

CS&E, JSSSTU, Mysore

Pareekshith Jain MP.

4th sem., 'C' sec.

Roll no.: 23, USN: 01JST18CS075

CS&E, JSSSTU, Mysore

Arya S Gangadkar.

4th sem., 'C' sec.

Roll no.: 10, USN: 01JST18CS017

CS&E, JSSSTU, Mysore

Submitted to:

Dr. Manimala S.

Professor,

CS&E, JSSSTU, Mysore

Abstract:

This document shows how to implement the "Electricity Bill with Client Server Architecture" using multithreads. The computer language used to implement this is 'Python'. The program uses 'threads' available in Python threading module. In order to synchronize the thread processes, the program uses 'lock'.

Table of Content:

1. Introduction

A. Client Server Architecture :

Client Server Architecture is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. This type of architecture has one or more client computers connected to a central server over a network or internet connection. This system shares computing resources. Client/server architecture is also known as a networking computing model or client/server network because all the requests and services are delivered over a network

B. The Purpose of Client/Server Architecture

We are in an era where information technology plays a critical role in business applications, considered as an area an organization would highly invest in order to widen the opportunities available to compete the global market. “A competitive global economy will ensure obsolescence and obscurity to those who cannot or are unwilling to compete”(Client/Server Architecture,2011), according to this statement it’s necessary for organizations sustain its market position by reengineering prevailing organizational structures and business practices to achieve their business goals. In short it’s a basic need to evolve with the change of technological aspects. Therefore organizations should undergo a mechanism to retrieve and process its corporate data to make business procedures more efficient to excel or to survive in the global market. The client/server model brings out a logical perspective of distributed corporative processing where a server handles and processes all client requests. This can be also viewed as a revolutionary milestone to the data processing industry. “Client/server computing is the most effective source for the tools that empower employees with authority and responsibility.”(Client/Server Architecture,2011) “Workstation power, workgroup empowerment, preservation of existing investments, remote network management, and market-driven business are the forces creating the need for client/server computing”. (Client/Server Architecture, 2011) Client/server computing has a vast progression in the computer industry leaving any area or corner untouched. Often hybrid skills are required for the development of client/server applications including database design, transaction processing, communication skills, graphical user interface design and development etc. Advanced applications require expertise of distributed objects and component infrastructures. Most commonly found client/server strategy today is PC LAN implementation optimized for the usage of group/batch. This has basically given threshold to many new distributed enterprises as it eliminates host-centric computing.

C. Characteristics of a Client-Server Architecture

- Client and server machines need different amount of hardware and software resources.
- Client and server machines may belong to different vendors.
- Horizontal scalability (increase of the client machines) and vertical scalability (migration to a more powerful server or to a multi-server solution)
- A client or server application interacts directly with a transport layer protocol to establish communication and to send or receive information.
- The transport protocol then uses lower layer protocols to send or receive individual messages. Thus, a computer needs a complete stack of protocols to run either a client or a server.
- A single server-class computer can offer multiple services at the same time; a separate server program is needed for each service.

The main aim of this project is to accept multiple client request and compute amount to be paid for each client concurrently. For this purpose we use concept called socket programming in Python. There's a lot of low-level stuff that needs to happen for these things to work but the Python socket module takes care of all of that, making network programming very easy for programmers.

D. Why to use threads in network programming?

The reason is simple, we don't want only a single client to connect to server at a particular time but many clients simultaneously. We want our architecture to support multiple clients at the same time. For this reason, we must use threads on server side so that whenever a client request comes, a separate thread can be assigned for handling each request.

Let us take an example, suppose a server is located at a place, say X. Being a generic server, it does not serve any particular client, rather to a whole set of generic clients. Also suppose at a particular time, two requests arrives at the server. With our basic server-client program, the request which comes even a nano-second first would be able to connect to the server and the other request would be rejected as no mechanism is provided for handling multiple requests simultaneously. To overcome this problem, we use threading in network programming.

2. Implementation

A. Socket API Overview

Python's socket module provides an interface to the Berkeley sockets API. This is the module that we'll use.

The primary socket API functions and methods in this module that we use are:

- socket()
- bind()
- listen()
- accept()
- connect()
- send()
- recv()
- close()

Python provides a convenient and consistent API that maps directly to these system calls, their C counterparts.

In our program we create a socket using socket() method which creates a TCP socket

B. TCP Sockets

Why should you use TCP? The Transmission Control Protocol (TCP):

- **Is reliable:** packets dropped in the network are detected and retransmitted by the sender.
- **Has in-order data delivery:** data is read by your application in the order it was written by the sender.

Networks are a best-effort delivery system. There's no guarantee that your data will reach its destination or that you'll receive what's been sent to you.

Network devices (for example, routers and switches), have finite bandwidth available and their own inherent system limitations. They have CPUs, memory, buses, and interface packet buffers, just like our clients and servers. TCP relieves you from having to worry about packet loss, data arriving out-of-order, and many other things that invariably happen when you're communicating across a network.

C. Server

```
import socket
import pandas as pd
from datetime import timedelta,date
import threading
#user defined exception class to handle exception raised
class InvalidUser(Exception):
    pass
class InvalidPower(Exception):
    pass
# Thread function to handle each client
def ClientHandler(conn,addr,lock):
    """
    curr: socket object
    addr: tuple containing host and port connected
    """
    data=pd.DataFrame()
    amount=25.0
    print('connected by',addr)
    conn.sendall("New User?[y/n]".encode('utf-8'))
    new=conn.recv(1024).decode('utf-8')
    if new=='y':
        try:
            conn.sendall("Enter ID".encode('utf-8'))
            id = conn.recv(1024).decode('utf-8')
            conn.sendall("Enter password".encode('utf-8'))
            passwd = conn.recv(1024).decode('utf-8')
            conn.sendall("Enter name".encode('utf-8'))
            name = conn.recv(1024).decode('utf-8')
            conn.sendall("Enter Last Meter Reading".encode('utf-8'))
            prevReading = float(conn.recv(1024).decode('utf-8'))
            new_row = pd.DataFrame({"User ID":id,"Password":passwd,"User Name":name,"Last
            meter reading":prevReading},index=[0])
            lock.acquire()
            df = pd.read_excel("F:\\OS Project\\Documentation\\Book1.xlsx")
```

```
            df = pd.concat([new_row, df]).reset_index(drop = True)
            df.to_excel("F:\\OS Project\\Documentation\\Book1.xlsx",index=False)
            lock.release()
            conn.sendall("Thankyou".encode('utf-8'))
        except:
            print("Error Occured")
        finally:
            print("Closing connection",addr)
            conn.close()
    else:
        try:
            df = pd.read_excel("F:\\OS Project\\Documentation\\Book1.xlsx")
            conn.sendall("Electricity bill calculation".encode('utf-8'))
            conn.sendall("Enter User ID".encode('utf-8'))
            id = conn.recv(1024).decode('utf-8')
            conn.sendall("Enter Password".encode('utf-8'))
            passwd = conn.recv(1024).decode('utf-8')
            if id in df["User ID"].values:
                data = df[df["User ID"]==id]
                if passwd not in data["Password"].values:
                    raise InvalidUser
            else:
                raise InvalidUser
            indx = data.index.values[0]
            name = data.at[indx,"User Name"]
            conn.sendall(("Welcome "+name).encode('utf-8'))
            prevReading = data.at[indx,"Last meter reading"]
            conn.sendall(("Previous reading ="+str(prevReading)).encode('utf-8'))
            conn.sendall("\nEnter current reading:".encode('utf-8'))
            currReading=float(conn.recv(1024).decode('utf-8'))
            power=currReading-prevReading
            if(power<0):
                raise InvalidPower
            if power<100:
                amount+=(power*1.5)
            elif power<200:
```

```
            amount=amount+(100*1.5)+(power-100)*2.5
        elif power<500:
            amount=amount+(100*1.5)+(100*2.5)+(power-200)*3.5
        else:
            amount=amount+(100*1.5)+(100*2.5)+(300*3.5)+(power-500)*4
        conn.sendall(("Amount is "+str(round(amount,2))).encode('utf-8'))
        conn.sendall(("\nAmount to be paid within   "+str((date.today()+
        timedelta(days=10)))).encode('utf-8'))
        conn.sendall("\nUpating in DataBase..".encode('utf-8'))
        lock.acquire()
        df = pd.read_excel("F:\\OS Project\\Documentation\\Book1.xlsx")
        data = df[df["User ID"]==id]
        indx = data.index.values[0]
        df.at[indx,"Last meter reading"]=currReading
        df.to_excel("F:\\OS Project\\Documentation\\Book1.xlsx",index=False)
        lock.release()
        conn.sendall("Updated".encode('utf-8'))
        conn.sendall("Thankyou".encode('utf-8'))
    except InvalidUser:
        conn.sendall("Invalid user".encode('utf-8'))
    except InvalidPower:
        conn.sendall("Invalid reading".encode('utf-8'))
    finally:
        print("closing connection",addr)
        conn.close()

# Setup server socket
HOST = '127.0.0.1'
PORT = 5000
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((HOST,PORT))
# acquire lock for synchronization
lock = threading.Lock()
print("Server running")
try:
```

```
    s.listen()
    while True:
        conn, addr = s.accept()
        threading.Thread(target=ClientHandler,args = (conn, addr, lock)).start()
except:
    print("Closing connection")
    s.close()
```

Let's walk through each API call and see what's happening.

socket.socket() creates a socket object. The arguments passed to socket() specify the address family and socket type. AF_INET is the Internet address family for IPv4. SOCK_STREAM is the socket type for TCP, the protocol that will be used to transport our messages in the network.

bind() is used to associate the socket with a specific network interface and port number.

```
HOST = '127.0.0.1'
PORT = 5000
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((HOST,PORT))
lock = threading.Lock()
```

The values passed to bind() depend on the address family of the socket. In this example, we're using socket.AF_INET (IPv4). So it expects a 2-tuple: (host, port). host can be a hostname, IP address, or empty string. If an IP address is used, host should be an IPv4-formatted address string. The IP address 127.0.0.1 is the standard IPv4 address for the loopback interface, so only processes on the host will be able to connect to the server. If you pass an empty string, the server will accept connections on all available IPv4 interfaces.

port should be an integer from 1-65535 (0 is reserved)

s.setsockopt() is used to make socket listen to multiple client request and threading.lock() is used to obtain a lock object used for synchronization

```
s.listen()
conn, addr = s.accept()
```

listen() enables a server to accept() connections. It makes it a “listening” socket. accept() blocks and waits for an incoming connection. When a client connects, it returns a new socket object representing the connection and a tuple holding the address of the client. The tuple will contain (host, port) for IPv4 connections.after getting conn object from accept(), it is used to communicate with client. conn.recv() is used to receive the data from client and conn.sendall() is used to send data to client. These uses bytes to send and receive data and we use UTF-8 as an encoding scheme.

After accepting the client we create a thread for that client using Pythons threading module. We use Excel as a database to store the uaser information and use pandas dataframe to manipulate DATA.

And finally we close connection using close() method.

D. Client

```
import socket
HOST = '127.0.0.1'
PORT = 5000
try:
    s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    s.connect((HOST,PORT))
    print("Connected\n")
    print(s.recv(1024).decode('utf-8'))
    new=input()
    s.sendall(new.encode('utf-8'))
    if new == 'y':
        print(s.recv(1024).decode('utf-8'))
        s.sendall(input().encode('utf-8'))
        print(s.recv(1024).decode('utf-8'))
```

```
        s.sendall(input().encode('utf-8'))
        print(s.recv(1024).decode('utf-8'))
        s.sendall(input().encode('utf-8'))
        print(s.recv(1024).decode('utf-8'))
        s.sendall(input().encode('utf-8'))
        print(s.recv(1024).decode('utf-8'))
    else:
        print(s.recv(1024).decode('utf-8'))
        print(s.recv(1024).decode('utf-8'))
        s.sendall(input().encode('utf-8'))
        print(s.recv(1024).decode('utf-8'))
        s.sendall(input().encode('utf-8'))
        print(s.recv(1024).decode('utf-8'))
        print(s.recv(1024).decode('utf-8'))
        s.sendall(input().encode('utf-8'))
        print(s.recv(1024).decode('utf-8'))
        print(s.recv(1024).decode('utf-8'))
        print(s.recv(1024).decode('utf-8'))
except :
    print("Error occured")

finally :
    print("Closing connection")
    s.close()
```

In comparison to the server, the client is pretty simple. It creates a socket object, connects to the server and calls s.sendall() to send its message. Lastly, it calls s.recv() to read the server's reply and then prints it.

3. Testing

Save the server program as Server.py and client program as Client.py. Open a terminal or command prompt, navigate to the directory that contains your scripts, and run the server:

```
$ ./echo-server.py
```

Your terminal will appear to hang. That's because the server is blocked (suspended) in a call:

```
conn, addr = s.accept()
```

It's waiting for a client connection. Now open another terminal window or command prompt and run the client:

```
$ ./echo-client.py
```

4. Output

Server Side:

```
 runfile('C:/Users/Kartik/Server.py', wdir='C:/Users/Kartik')
Server running
connected by ('127.0.0.1', 57259)
connected by ('127.0.0.1', 57268)
Closing connection ('127.0.0.1', 57259)
closing connection ('127.0.0.1', 57268)
```

Client 1:

```
runfile('C:/Users/Kartik/Client.py', wdir='C:/Users/Kartik')
Connected

New User?[y/n]
```

```
y
Enter ID

C12345
Enter password

BeHumble
Enter name

Rickyticky Bobbywobby
Enter Last Meter Reading

1235.3
Thankyou
Closing connection
```

Client 2:

```
 runfile('C:/Users/Kartik/Client.py', wdir='C:/Users/Kartik')
Connected

New User?[y/n]

n
Electricity bill calculation
Enter User ID

C14521
Enter Password

humble
Welcome Dr. Shrimp Portarico
Previous reading =251.32
Enter current reading:

299.3
```

```
Amount is 96.97

Amount to be paid within 2020-05-11
Upating in DataBase..
Updated
Closing connection
```

5. Conclusion

The Client-Server network model provides important services to the network safely and securely, it also allows the convenience of allowing the users to work on their own workstation machine. However, this network model can be very expensive, not only because the software can be expensive, but you also must provide adequate hardware for both the servers and the individual workstation machines, which can become very expensive with revolving hardware updates.

If you have the funds to implement this type of network, the return on the investment is great, and you will have the knowledge that your network is well secured and archived

6. Reference

- Abraham_Silberschatz (Yale University), Peter Baer Galvin (Pluribus Networks), Greg Gagne (Westminster College), Ninth edition -Operating_System_Concepts
- en.wikipedia.org/
- www.geeksforgeeks.org/
- www.realpython.com
- docs.python.org
- stackoverflow.com