

Department of Computer Engineering

Academic Term : January - May 2024

Class: B.E. (Computer) Semester VIII

Division: B

Subject Name: Distributed Systems CSC 801

Name	Riya Rajesh Patil
Roll No.	9282
Experiment No.	9
Experiment Title	Implementation of Stateful/Stateless servers

Aim - Implementation of Stateful/Stateless servers

STATEFUL SERVER

server.py

```
import socket
```

```
import time
```

```
HOST = '127.0.0.1' # Symbolic name meaning all available interfaces
```

```
PORT = 8888 # Arbitrary non-privileged port
```

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

```
    s.bind((HOST, PORT))
```

```
    s.listen(1)
```

```
    print('Server started and listening on port', PORT)
```

```
    while True:
```

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

```
        print('Client connected:', addr)
```

```

data = b"
start_time = time.time()
while True:
    chunk = conn.recv(1024)
    data += chunk
    if len(chunk) < 1024:
        break
num1, num2 = map(int, data.decode().split())
result = num1 * num2
response_time = time.time() - start_time
print('Received:', num1, num2)
print('Result:', result)
print('Response time:', response_time)
conn.sendall(str(result).encode())

```

client.py

```

import socket
import time

```

```

HOST = 'localhost' # The server's hostname or IP address
PORT = 8888 # The port used by the server

```

```

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    print("Calculating product")
    num1 = 22
    num2 = 9
    start_time = time.time()
    s.sendall(f'{num1} {num2}'.encode())

```

```

data = b"
while True:
    chunk = s.recv(1024)
    data += chunk
    if len(chunk) < 1024:
        break
response_time = time.time() - start_time
result = int(data.decode())
print('Received:', result)

```

```

(base) riyapatil@Riyas-Air DC_9 % python server.py
Server started and listening on port 8888
Client connected: ('127.0.0.1', 50557)
Received: 22 9
Result: 198
Response time: 3.814697265625e-05

```

```

Received: 198
(base) riyapatil@Riyas-Air DC_9 % python client.py
Calculating product
Received: 198
(base) riyapatil@Riyas-Air DC_9 %

```

STATELESS SERVER

client.py

```

import socket
import time

```

HOST = '127.0.0.1' # The server's hostname or IP address

PORT = 8888 # The port used by the server

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

```
    s.connect((HOST, PORT))
```

```
    print("Calculating product")
```

```
    num1 = 5
```

```
    num2 = 6
```

```
    start_time = time.time()
```

```
    s.sendall(f'{num1} {num2}'.encode())
```

```
    data = b''
```

```
    while True:
```

```
        chunk = s.recv(1024)
```

```
        data += chunk
```

```
        if len(chunk) < 1024:
```

```
            break
```

```
    response_time = time.time() - start_time
```

```
    result = int(data.decode())
```

```
    print('Received:', response_time)
```

server.py

```
import socket
```

```
import time
```

```
HOST = '127.0.0.1' # Symbolic name meaning all available interfaces
```

```
PORT = 8888 # Arbitrary non-privileged port
```

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

```
    s.bind((HOST, PORT))
```

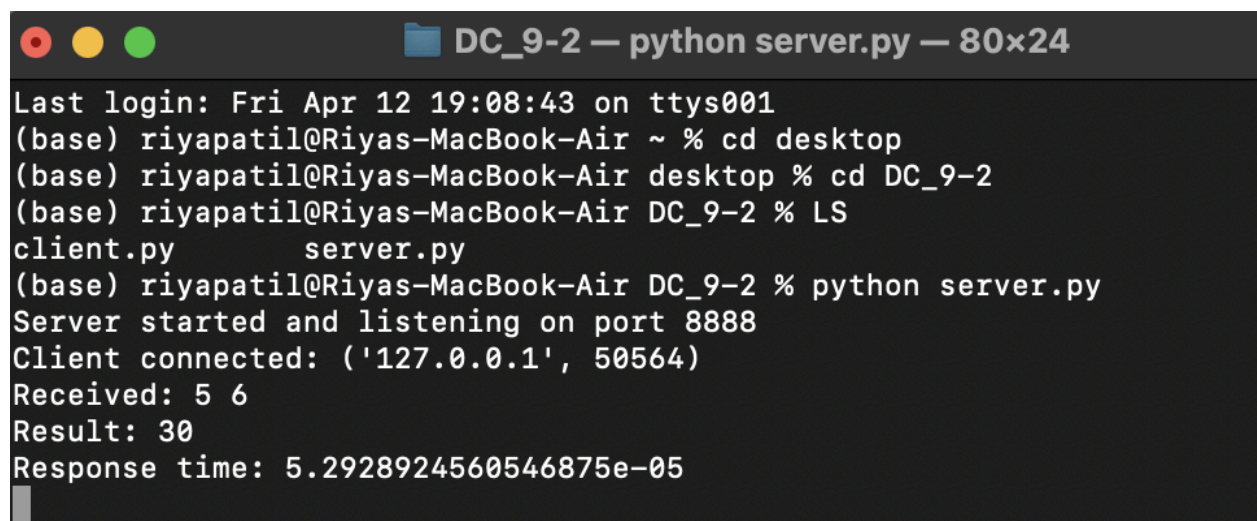
```
    s.listen(1)
```

```

print('Server started and listening on port', PORT)

while True:
    conn, addr = s.accept()
    print('Client connected:', addr)
    with conn:
        data = conn.recv(1024)
        if not data:
            continue
        start_time = time.time()
        num1, num2 = map(int, data.decode().split())
        result = num1 * num2
        response_time = time.time() - start_time
        print('Received:', num1, num2)
        print('Result:', result)
        print('Response time:', response_time)
        conn.sendall(str(result).encode())

```



A terminal window titled "DC_9-2 — python server.py — 80x24" showing the execution of a Python server script. The terminal output includes the login time, directory navigation, file listing, and the server's response to a client connection.

```

Last login: Fri Apr 12 19:08:43 on ttys001
(base) riyapatil@Riyas-MacBook-Air ~ % cd desktop
(base) riyapatil@Riyas-MacBook-Air desktop % cd DC_9-2
(base) riyapatil@Riyas-MacBook-Air DC_9-2 % LS
client.py      server.py
(base) riyapatil@Riyas-MacBook-Air DC_9-2 % python server.py
Server started and listening on port 8888
Client connected: ('127.0.0.1', 50564)
Received: 5 6
Result: 30
Response time: 5.2928924560546875e-05

```

```
DC_9-2 — -zsh — 80x24
Last login: Fri Apr 12 19:29:35 on ttys000
(base) riyapatil@Riyas-MacBook-Air ~ % cd desktop

(base) riyapatil@Riyas-MacBook-Air desktop % cd DC_9-2
(base) riyapatil@Riyas-MacBook-Air DC_9-2 % ls
client.py      server.py
(base) riyapatil@Riyas-MacBook-Air DC_9-2 % python client.py
Calculating product
Received: 0.00035500526428222656
(base) riyapatil@Riyas-MacBook-Air DC_9-2 %
```

Conclusion -

1. Implemented Stateful and Stateless Server
2. Compared the performance of stateful and stateless servers in handling client requests.
3. The response times of the stateless server were consistently faster and more stable compared to the stateful server.
4. Since the stateful server needs to keep track of client sessions, it requires more resources and processing power, which can result in slower response times and higher variability.
5. It is important to consider the requirements of the application and choose the appropriate server architecture based on those requirements.

Postlab -

1. What are the design issues of servers?

Designing servers involves several important considerations to ensure they meet the requirements of the system and provide efficient and reliable services. Some of the key design issues of servers include:

1. Scalability: Servers should be designed to handle increasing loads and accommodate growing numbers of clients without significant degradation in performance. This often involves implementing scalable architectures and employing techniques such as load balancing and clustering.
2. Performance: Servers need to deliver high performance to meet the response time and throughput requirements of the system. Design decisions related to hardware

selection, software architecture, and optimization techniques impact the overall performance of the server.

3. Reliability: Servers must be reliable to ensure continuous operation and minimize downtime. Redundancy, fault tolerance mechanisms, error handling, and recovery strategies are essential design considerations to improve reliability.

4. Security: Security is paramount for servers, especially when handling sensitive data or providing critical services. Designing servers with robust security features such as authentication, encryption, access control, and intrusion detection helps protect against unauthorized access and malicious attacks.

5. Maintainability: Servers should be designed for ease of maintenance and management to facilitate troubleshooting, updates, and upgrades. Modular design, documentation, logging, and monitoring capabilities contribute to the maintainability of servers.

2. Difference between stateful & stateless servers.

Stateful Servers:

1. Maintain Client State: Stateful servers keep track of client sessions and store information about the state of ongoing interactions. This allows them to remember previous interactions and provide personalized responses based on the client's history.

2. Session Management: They manage session identifiers or tokens to associate incoming requests with specific client sessions. This enables continuity of client-state across multiple requests.

3. Resource Utilization: Stateful servers typically consume more resources (such as memory and processing power) to manage client state and session information.

4. Scalability Challenges: Scaling stateful servers can be more complex because session state needs to be synchronized across multiple server instances to ensure consistent client experiences.

5. Enhanced Functionality: Stateful servers are often used in applications that require maintaining complex session data, such as e-commerce platforms, online gaming, and real-time collaboration tools.

Stateless Servers:

1. No Client State: Stateless servers do not store any client-specific information between requests. Each request is treated independently, without any memory of previous interactions.

2. Simplified Architecture: Stateless servers have a simpler architecture compared to stateful servers because they do not need to manage session state or track client sessions.

3. **Reduced Resource Consumption:** Stateless servers typically consume fewer resources because they do not maintain client state. They can handle a larger number of concurrent requests with lower memory overhead.
4. **Scalability Benefits:** Stateless servers are inherently more scalable because they can easily distribute incoming requests across multiple server instances without the need for session synchronization.
5. **Stateless Protocols:** Many web-based protocols, such as HTTP, are inherently stateless, making them well-suited for stateless server architectures. Each HTTP request contains all the information needed for processing, without relying on session state.