Extra Problems Report Tutorial 1

Srivathsa vamsi Chaturvedula
22110260

February 7, 2025

1 Problem 1: Client-Server Task Processing Program

1.1 Overview

This program consists of a client and three different server designs that process tasks sent by the client.

1.2 Client Side Program

- **IP Address Retrieval**
  This function uses the `netifaces` library to obtain the IP address of a specified network interface.
- **Socket Creation and Connection**
  This code creates a TCP socket and establishes a connection to the server.
- **User Interface Loop**
  This loop presents a menu to the user and captures their choice.
- **Request Formatting and Sending**
  The client formats the request and sends it to the server.
- **Response Handling**
  The client waits for and receives the server's response.

```python
import socket
import netifaces as ni

def get_ip_address(interface):
    return ni.ifaddresses(interface)[ni.AF_INET][0]['addr']

def run_client(server_interface='eth0', server_port=65432):
    server_host = get_ip_address(server_interface)
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((server_host, server_port))
        print(f"Connected to server at {server_host}:{server_port}")
        while True:
            print("\nChoose an action:")
            print("1. Swap case of string")
            print("2. Evaluate arithmetic expression")
            print("3. Reverse string")
            print("4. Exit")
            choice = input("Enter choice (1-4): ")

            if choice == '4':
                break

            if choice in ('1', '2', '3'):
                data = input("Enter input: ")
                s.sendall(f"{choice}|{data}".encode())
                response = s.recv(1024).decode()
                print("Result:", response)
            else:
                print("Invalid choice")

if __name__ == "__main__":
    run_client(server_interface='eth0')
```

Figure 1: client side program

1.3 Server Side Program

a) Network Interface Configuration

The server dynamically retrieves its IP address from a specified network interface.

```python
def get_ip_address(interface='eth0'):
    """Retrieve the IP address of the specified network interface."""
    return ni.ifaddresses(interface)[ni.AF_INET][0]['addr']
```

b) Request Processing Logic

The function processes incoming data based on the operation type.

```python
def process_request(data):
    """Process incoming client request and return appropriate response."""
    try:
        action, payload = data.split('|', 1)
        if action == '1':
            return payload.swapcase()
        elif action == '2':
            return str(eval(payload))
        elif action == '3':
            return payload[::-1]
        return "Invalid choice"
    except Exception as e:
        return f"Error: {str(e)}"
```

c) Client Connection Handler

The function manages client connections and processes requests.

```python
def handle_client(conn, addr):
    """Handle communication with a connected client."""
    print(f"[{datetime.now().strftime('%H:%M:%S')}] Connection from {addr}")
    with conn:
        while True:
            data = conn.recv(1024).decode()
            if not data:
                break
            response = process_request(data)
            conn.sendall(response.encode())
    print(f"[{datetime.now().strftime('%H:%M:%S')}] Client {addr} disconnected")
```

d) Server Architectures

- **Single-Process Server:** Handles one client at a time.

server_single.py

```python
import socket
from utils import get_ip_address, handle_client

def single_process_server(interface='eth0', port=65432):
    """Start a single-process server that handles one client at a time."""
    host = get_ip_address(interface)
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
        server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        server_socket.bind((host, port))
        server_socket.listen()
        print(f"Single-Process Server running on {host}:{port}")

        while True:
            conn, addr = server_socket.accept()
            handle_client(conn, addr)

if __name__ == "__main__":
    single_process_server()
```

- **Multi-Process Server:** Uses `os.fork()` to handle multiple clients concurrently.

server_multi_process.py

```python
import socket
import os
from utils import get_ip_address, handle_client

def multi_process_server(interface='eth0', port=65432):
    """Start a multi-process server where each client is handled in a separate process."""
    host = get_ip_address(interface)
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
        server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        server_socket.bind((host, port))
        server_socket.listen()
        print(f"Multi-Process Server running on {host}:{port}")

        while True:
            conn, addr = server_socket.accept()
            pid = os.fork()

            if pid == 0:  # Child process
                server_socket.close()
                handle_client(conn, addr)
                os._exit(0)
            else:  # Parent process
                conn.close()

if __name__ == "__main__":
    multi_process_server()
```

- **Multi-Threaded Server:** Uses `threading.Thread()` to manage multiple clients.

server_multi_thread.py

```python
import socket
import threading
from utils import get_ip_address, handle_client

def multi_threaded_server(interface='eth0', port=65432):
    """Start a multi-threaded server where each client is handled in a separate thread."""
    host = get_ip_address(interface)
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
        server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        server_socket.bind((host, port))
        server_socket.listen()
        print(f"Multi-Threaded Server running on {host}:{port}")

        while True:
            conn, addr = server_socket.accept()
            thread = threading.Thread(target=handle_client, args=(conn, addr))
            thread.start()

if __name__ == "__main__":
    multi_threaded_server()
```

1.4 Testing of Programs

Execution is demonstrated using two terminals of Ubuntu WSL machine. The server runs with different modes:

- Single-Process Server: Only one client is served at a time.

```
(base) vamix@winSriva:~$ cd cntut1_coding/
(base) vamix@winSriva:~/cntut1_coding$ cd P1
(base) vamix@winSriva:~/cntut1_coding/P1$ conda activate cntut
(cntut) vamix@winSriva:~/cntut1_coding/P1$ python3 server_single.py
Single-Process Server running on 172.30.240.144:65432
[18:32:30] Connection from ('172.30.240.144', 44686)
```

```
(base) vamix@winSriva:~$ cd cntut1_coding/
(base) vamix@winSriva:~/cntut1_coding$ cd P1
(base) vamix@winSriva:~/cntut1_coding/P1$ conda activate cntu

EnvironmentNameNotFound: Could not find conda environment: cntu
You can list all discoverable environments with `conda info --envs`.


(base) vamix@winSriva:~/cntut1_coding/P1$ conda activate cntut
(cntut) vamix@winSriva:~/cntut1_coding/P1$ python3 client.py
Connected to server at 172.30.240.144:65432

1. Change case
2. Evaluate expression
3. Reverse string
4. Exit
Enter choice (1-4): 3
Enter input: vamsi
Result: ismav

1. Change case
2. Evaluate expression
3. Reverse string
4. Exit
Enter choice (1-4):
```

- Multi-Process Server: Multiple clients are served using new processes.

```
(cntut) vamix@winSriva:~/cntut1_coding/P1$ python3 server_multi_process.py
Multi-Process Server running on 172.30.240.144:65432
[18:50:47] Connection from ('172.30.240.144', 40868)
[18:51:26] Connection from ('172.30.240.144', 40820)
```

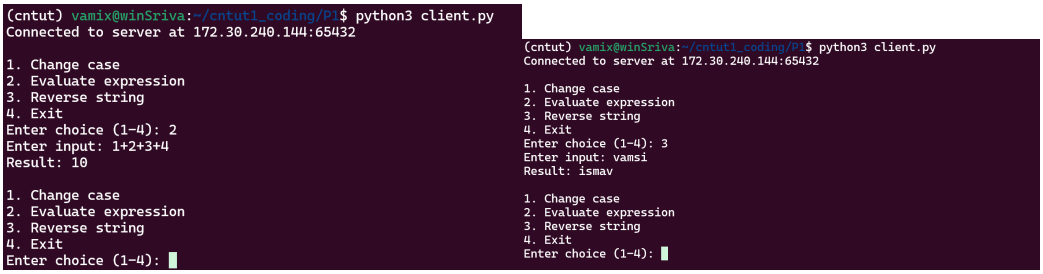

Figure 2: running 2 clients on 2 different terminals of a WSL machine

- Multi-Threaded Server: Multiple clients are served using threads.(testing of this program would be same as the multi process one)

2 Problem 2: Client-Server Benchmarking Task

2.1 Overview

This task benchmarks a client-server architecture where a client sends a string, and the server responds after a 3-second delay. So made modifications in client.py accordingly.

2.2 Benchmark Tests

Execution time is measured with varying clients, and results are plotted.

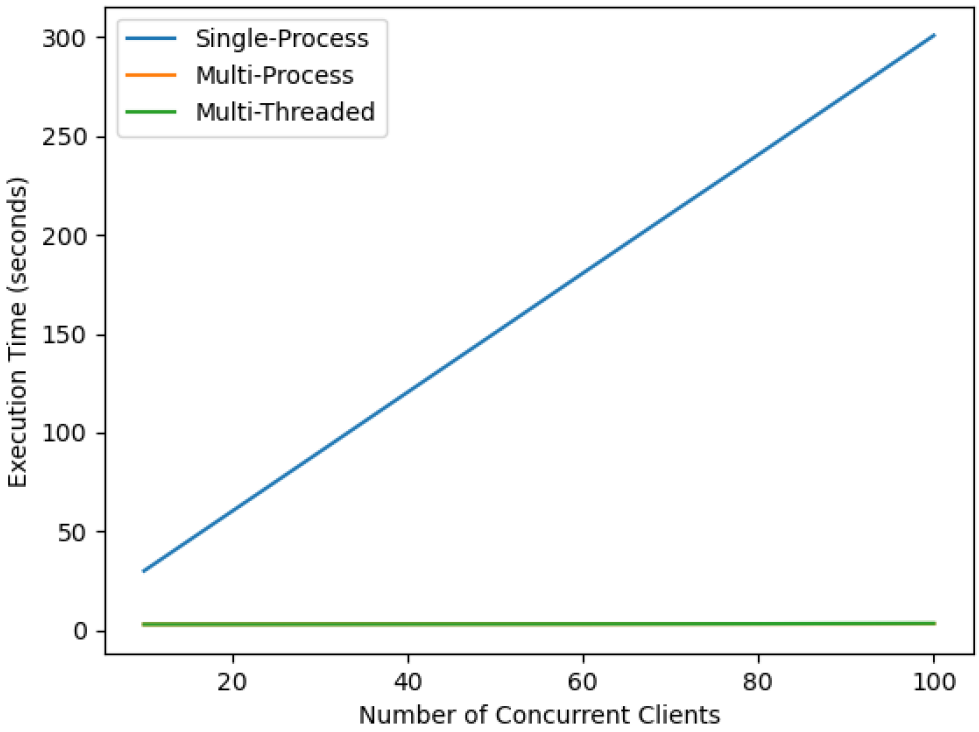

Figure 3: Number of concurrent clients vs Execution time

The results show that Multi-Process and Multi-Threaded servers handle concurrent clients efficiently.