Number Guessing Game

Aaron Jones

4/17/2024

CYSE 250

The field of web programming is broad and crucial for the development of distributed applications. The goal of this project was to create a number guessing game that uses a client-server architecture to facilitate real-time communication between multiple clients and a central server. The main challenge was to design and implement a system that could effectively manage simultaneous connections and gaming sessions while providing an immersive user experience. The goal of the game is simple: customers guess a number and the server gives hints until the correct number is guessed. But the underlying network communication and concurrency management presented complex problems to solve, such as handling multiple client connections, synchronizing game state, and ensuring server responsiveness and scalability.

This project was developed on a Dell computer that provided the computing power and networking capabilities needed to host and test server-client interactions. System specifications included an Intel Core i5 processor and 8GB of RAM, which ensures that the server can handle multiple client connections without significant performance degradation.

The software was carefully selected to meet the requirements of the project. The operating system was Windows 10 version 1909, which offered a stable and familiar development environment. The project was written in Python 3.12.2, the latest version at the time, which included improvements to make the development process easier, especially in web programming. Visual Studio Code version 3.12 was chosen as the IDE because of its extensive Python support, including debugging tools, and a user-friendly interface that sped up the coding and testing phases.

The software used to implement the project The implementation of the Number Guessing game was based on the Python standard library, especially the networking module and the concurrent chaining module. The server script used socket programming to listen for incoming connections and created a new thread for each client to manage multiple simultaneous game sessions. The client script is designed to connect to the server, send guesses and receive feedback. Both scripts included exception handling to handle potential network errors and ensure a robust application.

Results and Discussions the Number Guessing Game was successfully implemented and met the project's functional requirements. The server was capable of handling multiple clients concurrently, and users could interact with the game as intended. During testing, it was observed that the server-maintained performance and responsiveness even as the number of clients increased, validating the effectiveness of the threading model used.

```
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

server_socket.bind(('localhost', 12345))

server_socket.listen()
print("Number Guessing Game server is running...")
Number Guessing Game server is running...
while True:
    client_socket, client_address = server_socket.accept()
    print(f"{client_address} has connected.")
    thread = threading.Thread(target=client_handler, args=(client_socket, client_address))
    thread.start()
    client_threads.append(thread)


('127.0.0.1', 52396) has connected.
('127.0.0.1', 52496) has connected.
```

```
IDLE Shell 3.12.2 - C:/Users/johnt/OneDrive/Desktop/milestone22.py (3.12.2)
Edit  Shell  Debug  Options  Window  Help
2
Too low!
Enter your guess (1-100): 45
2
Too low!
Enter your guess (1-100): 57
2
Too high!
Enter your guess (1-100): 56
2
Too high!
Enter your guess (1-100): 48
2
Too high!
Enter your guess (1-100): 47
2
Correct! You guessed it in 16 attempts.
```

The project's success in creating a number-guessing game using a server-client model demonstrated the practical application of web programming concepts in Python. This provided valuable insight into the complexity of managing concurrent connections and the importance of thorough testing of distributed systems. While the current implementation has met the project's goals, future improvements may include a graphical user interface for customers, improved gameplay features, and scalability optimizations.

Appendix:

Server

```
import socket
import threading
import random

games = {}

client_threads = []

def client_handler(client_socket, client_address):
    secret_number = random.randint(1, 100)
    games[client_address] = secret_number
    client_socket.send("Welcome to the Number Guessing Game! Guess a number between 1 and 100.".encode('utf-8'))
    attempts = 0
    while True:
        try:
            guess = client_socket.recv(1024).decode('utf-8')
            if guess:
                attempts += 1
                guess = int(guess)
                log_guess(client_address, guess)
                if guess < secret_number:
                    client_socket.send("Too low!".encode('utf-8'))
                elif guess > secret_number:
                    client_socket.send("Too high!".encode('utf-8'))
                else:
                    client_socket.send(f"Correct! You guessed it in {attempts} attempts.".encode('utf-8'))
                    break
        except (socket.error, ValueError):
            break
    del games[client_address]
    client_socket.close()

def log_guess(client_address, guess):
    with open("guess_log.txt", "a") as file:
        file.write(f"{client_address}: {guess}\n")

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

server_socket.bind(('localhost', 12345))

server_socket.listen()
print("Number Guessing Game server is running...")
Number Guessing Game server is running...
while True:
    client_socket, client_address = server_socket.accept()
    print(f"{client_address} has connected.")
    thread = threading.Thread(target=client_handler, args=(client_socket, client_address))
    thread.start()
    client_threads.append(thread)
```

Client

```
import socket
import threading
```

Create a TCP/IP socket

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(('localhost', 12345))
```

Connect socket to server

```
welcome_message = client_socket.recv(1024).decode('utf-8')
print(welcome_message)
Welcome to the Number Guessing Game! Guess a number between 1 and 100.
```

Main loop for the client to send guesses to server

```
while True:
    guess = input('Enter your guess (1-100): ')
    client_socket.send(guess.encode('utf-8'))
    response = client_socket.recv(1024).decode('utf-8')
    print(response)
    if response.startswith("Correct!"):
        break
```

Exit loop if the guess was correct