

Design Document for Multi-Threaded Server and Client

Luke Pepin, CSE 3300, Tuesday October 22, 2024

Description

Overall Program Design: The basic and multi-threaded server and client program is designed to handle wildcard queries for English words stored in a word list. The server listens for incoming connections from clients, processes their queries, and returns matching words. The client sends queries to the server and displays the results. The server can handle multiple clients simultaneously by spawning a new thread for each client connection. Both pairs of programs are modified versions of the starter code `thread-server.py` as provided in the assignment folder.

How It Works:

The server initializes a TCP socket, binds it to a specified port, and listens for connections. Upon accepting a connection, it spawns a new thread to handle the client. Each thread processes the client's query by converting the wildcard pattern to a regular expression, searching the word list for matches, and sending the results back to the client. The server responds with a status code (200 OK or 404 Not Found), the number of matching words, and the list of matching words. The client initializes a TCP socket, connects to the server, prompts the user to enter a wildcard query, sends the query to the server, receives the response, and displays the results.

The client initializes a TCP socket, connects to the server, prompts the user to enter a wildcard query, sends the query to the server, receives the response, and displays the results. (From both sets of Client/Server Pairs)

The application protocol involves the client sending a query string with the possibility of containing the wildcard pattern `?`, and the server responding with a status code, the number of matching words, and the list of matching words. The response is determined directly by three lines (modified for publishing):

- `response= 404 Not Found`
- `if length of matches != 0`
 - `response = 200 OK, length of matches, matches ...`

Tradeoffs

Design tradeoffs include choosing threading over forking due to its portability across different operating systems, including Windows. Threads share the same memory space, which simplifies the management of shared resources and reduces overhead compared to forking, where each process has its own memory space. Regular expressions were chosen for their flexibility and power in pattern matching, allowing for complex queries and efficient searching. Blocking I/O was selected for its simplicity, as each thread handles a client connection in a straightforward manner, avoiding the complexity and potential debugging challenges associated with non-blocking I/O.

Extensions

The two most beneficial extensions to the programs are keeping the client open until a 'quit' value is sent, allowing for continued searches without reconnecting, and improving error handling. Keeping the client open enhances user experience by enabling multiple queries in a single session, reducing the overhead of repeated connections. Improved error handling addresses common issues such as the `ConnectionResetError: [WinError 10054]`, which occurs when the connection is forcibly closed by the remote host. This can be mitigated by

Test Cases

1. Setup: The Server/Client are both executable codes and wait for the client query.
2. Basic Success: A basic query with no wildcards is successfully ran.
3. Wildcard Success: A wildcard query is successfully ran.
4. Multi Wildcard Success: A Multi Wildcard query is successfully ran
5. Invalid Query: An invalid query is ran displaying 404 Not Found.

Screenshots:

Basic Server/Client Screenshots



```

# basic_server.py
1 import socket
2
3 # listen on a non-reserved port number
4 port = 5000
5
6 sockobj = socket(AF_INET, SOCK_STREAM)
7 # make a TCP socket object
8 sockobj.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
9 # bind it to server port number
10 sockobj.bind((ipynote, port))
11 # allow up to 5 pending connects
12 sockobj.listen(5)
13
14 def now():
15     return time.time()
16
17 # current time on the server
18
19 def handleClient(connection):
20     # in spawned thread: reply
21     time.sleep(1)
22     # simulate a blocking activity
23     with open('wordlist.txt', 'r') as file:
24         wordlist = file.read().splitlines()
25
26     while True:
27         # read, write a client socket
28         data = connection.recv(1024)
29         if not data: break
30         query = data.decode().strip()
31         # Decode the data to a string
32         pattern = re.compile(query.replace('*', ''))
33         # Convert wildcard to regex
34         matches = [word for word in wordlist if pattern.search(word)]
35
36         response = "Add Not Found"
37         # Data response is not found
38         if len(matches) != 0:
39             # If matches exist 200 OK ...
40             response = f"200 OK (Number of matching words: {len(matches)})" + "\n".join(matches)
41
42 # main()
43
44 if __name__ == '__main__':
45     main()
46
47 # basic_client.py
48
49 # get socket constructor
50 import socket
51
52 # set localhost and port time as server
53 server_address = ('localhost', 5000)
54
55 # create a TCP/IP socket
56 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
57 sock.connect(server_address)
58
59 # send the wildcard query from the user
60 query = input("Enter your wildcard query (e.g., 'a*'): ")
61 print(f"Sending query: {query}")
62 sock.sendall(query.encode())
63
64 # send the query to the server
65
66 # receive the response (larger buffer for potentially non-
67 data = sock.recv(1024)
68 response = data.decode()
69
70 # Print the server's response
71 print(f"Received response: {response}")
72 sock.close()
73
74 # main()
75
76 if __name__ == '__main__':
77     main()
78
79 # ConnectionError: [Errno 10061] No connection could be made because the target machine actively refused it
80 PS C:\Users\Lake\PeppinOneDrive\Documents\Academics\VA1> python .\basic_client.py
81 Enter your wildcard query (e.g., 'a*'): f*
82 Sending query: f*
83 Received response:
84 200 OK
85 Number of matching words: 227
86 effectusio
87 aestetico
88 beefsteak
89 befit
90 befitting
91 benefit
92 benefits
93 buffet

```

Figure 3: Basic Success Wildcard

```

# basic_server.py
1 import socket
2
3 # listen on a non-reserved port number
4 port = 5000
5
6 sockobj = socket(AF_INET, SOCK_STREAM)
7 # make a TCP socket object
8 sockobj.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
9 # bind it to server port number
10 sockobj.bind((ipynote, port))
11 # allow up to 5 pending connects
12 sockobj.listen(5)
13
14 def now():
15     return time.time()
16
17 # current time on the server
18
19 def handleClient(connection):
20     # in spawned thread: reply
21     time.sleep(1)
22     # simulate a blocking activity
23     with open('wordlist.txt', 'r') as file:
24         wordlist = file.read().splitlines()
25
26     while True:
27         # read, write a client socket
28         data = connection.recv(1024)
29         if not data: break
30         query = data.decode().strip()
31         # Decode the data to a string
32         pattern = re.compile(query.replace('*', ''))
33         # Convert wildcard to regex
34         matches = [word for word in wordlist if pattern.search(word)]
35
36         response = "Add Not Found"
37         # Data response is not found
38         if len(matches) != 0:
39             # If matches exist 200 OK ...
40             response = f"200 OK (Number of matching words: {len(matches)})" + "\n".join(matches)
41
42 # main()
43
44 if __name__ == '__main__':
45     main()
46
47 # basic_client.py
48
49 # get socket constructor
50 import socket
51
52 # set localhost and port time as server
53 server_address = ('localhost', 5000)
54
55 # create a TCP/IP socket
56 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
57 sock.connect(server_address)
58
59 # send the wildcard query from the user
60 query = input("Enter your wildcard query (e.g., 'a*'): ")
61 print(f"Sending query: {query}")
62 sock.sendall(query.encode())
63
64 # send the query to the server
65
66 # receive the response (larger buffer for potentially non-
67 data = sock.recv(1024)
68 response = data.decode()
69
70 # Print the server's response
71 print(f"Received response: {response}")
72 sock.close()
73
74 # main()
75
76 if __name__ == '__main__':
77     main()
78
79 # ConnectionError: [Errno 10061] No connection could be made because the target machine actively refused it
80 PS C:\Users\Lake\PeppinOneDrive\Documents\Academics\VA1> python .\basic_client.py
81 Enter your wildcard query (e.g., 'a*'): t*
82 Sending query: t*
83 Received response:
84 200 OK
85 Number of matching words: 489
86 about
87 aboutthought
88 aboutthoughtful
89 aboutthoughtful
90 aboutthoughtful
91 aboutthoughtful
92 aboutthoughtful
93 aboutthoughtful
94 aboutthoughtful
95 aboutthoughtful
96 aboutthoughtful
97 aboutthoughtful
98 aboutthoughtful
99 aboutthoughtful
100 aboutthoughtful

```

Figure 4: Basic Success Multiple Wildcards

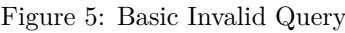
[illegible]

Image 5:

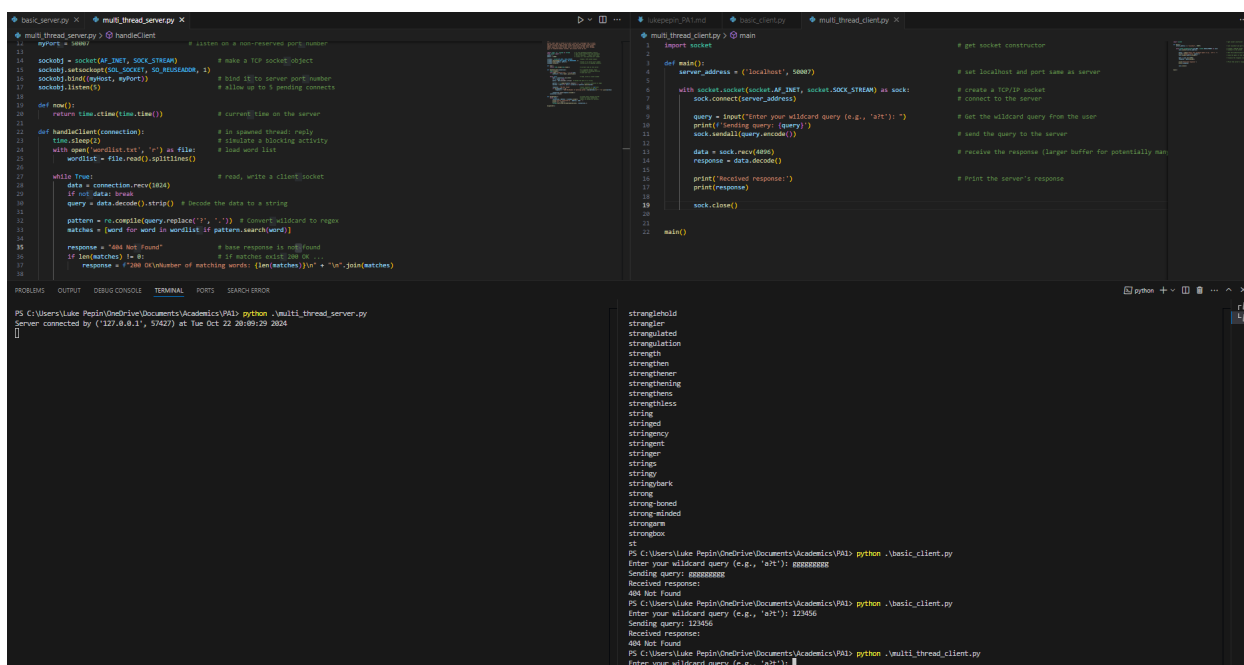


Figure 6: Multi-Thread Setup

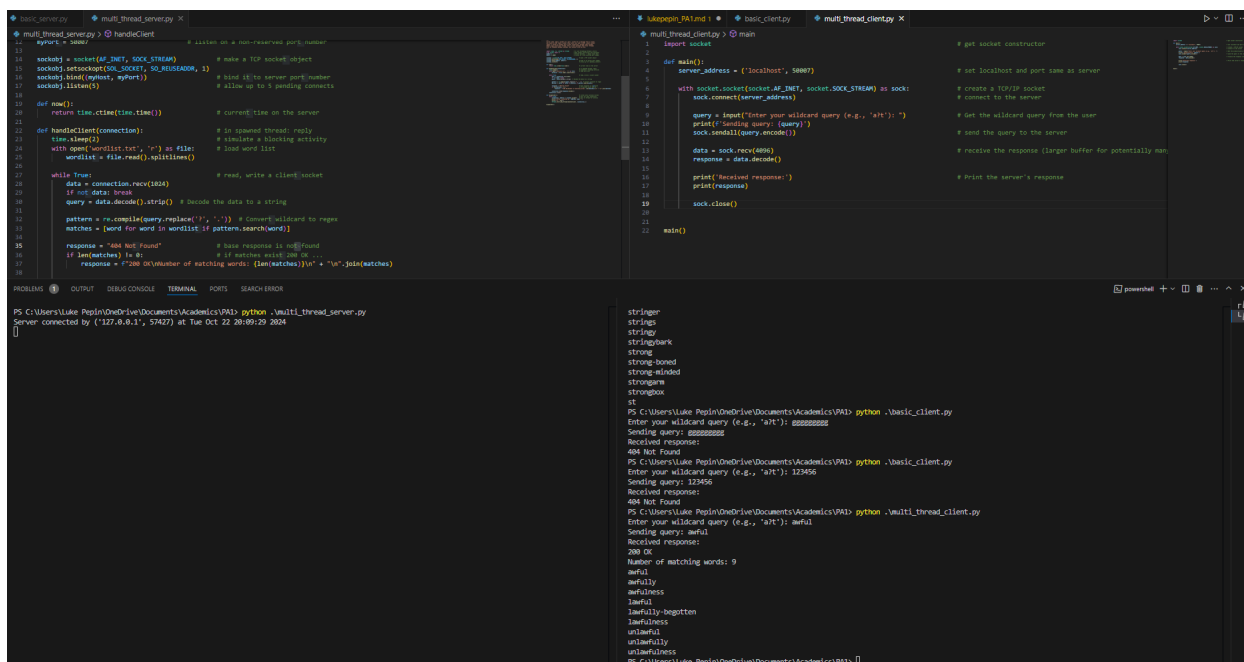


Figure 7: Multi-Thread Success No Wildcard

```

# multi_thread_server.py
import socket
import sys
import time

def handleClient(connection):
    # listen on a non-reserved port number
    # make a TCP socket object
    # bind it to the server port number
    # allow up to 5 pending connects
    # current time on the server
    # in spawned thread: reply
    # simulate a blocking activity
    # load word list
    # read, write a client socket
    # decode the data to a string
    # convert wildcard to regex
    # search for word in wordlist
    # base response is not found
    # if matches exist join them
    # response = "200 OK: Number of matching words: {len(matches)}" + " ".join(matches)

# multi_thread_client.py
import socket
import sys

def main():
    # get socket constructor
    # set localhost and port same as server
    # create a TCP socket
    # connect to the server
    # get the wildcard query from the user
    # send the query to the server
    # receive the response (larger buffer for potentially mem)
    # print the server's response
    # sock.close()
    main()

# Terminal Output
PS C:\Users\Luke Pepin\OneDrive\Documents\Academics\PA1> python .\multi_thread_server.py
Server connected by ('127.0.0.1', 57492) at Tue Oct 22 20:10:20 2024
Server connected by ('127.0.0.1', 57492) at Tue Oct 22 20:10:23 2024
Server connected by ('127.0.0.1', 57555) at Tue Oct 22 20:11:10 2024
Server connected by ('127.0.0.1', 57563) at Tue Oct 22 20:11:20 2024

vshillyshilly
PS C:\Users\Luke Pepin\OneDrive\Documents\Academics\PA1> python .\multi_thread_client.py
Enter your wildcard query (e.g., 'a*'): fantastic
Sending query: fantastic
Received response:
404 Not Found
PS C:\Users\Luke Pepin\OneDrive\Documents\Academics\PA1> python .\multi_thread_client.py
Enter your wildcard query (e.g., 'a*'): great
Sending query: great
Received response:
200 OK
Number of matching words: 23
egregious
egregious
allegretto
allegretto
egregious
great
great-sunt

```

Figure 8: Multi-Thread Success Wildcard

```

# multi_thread_server.py
import socket
import sys
import time

def handleClient(connection):
    # listen on a non-reserved port number
    # make a TCP socket object
    # bind it to the server port number
    # allow up to 5 pending connects
    # current time on the server
    # in spawned thread: reply
    # simulate a blocking activity
    # load word list
    # read, write a client socket
    # decode the data to a string
    # convert wildcard to regex
    # search for word in wordlist
    # base response is not found
    # if matches exist join them
    # response = "200 OK: Number of matching words: {len(matches)}" + " ".join(matches)

# multi_thread_client.py
import socket
import sys

def main():
    # get socket constructor
    # set localhost and port same as server
    # create a TCP socket
    # connect to the server
    # get the wildcard query from the user
    # send the query to the server
    # receive the response (larger buffer for potentially mem)
    # print the server's response
    # sock.close()
    main()

# Terminal Output
PS C:\Users\Luke Pepin\OneDrive\Documents\Academics\PA1> python .\multi_thread_server.py
Server connected by ('127.0.0.1', 57492) at Tue Oct 22 20:10:20 2024
Server connected by ('127.0.0.1', 57492) at Tue Oct 22 20:10:23 2024

unlawful
unlawful
unlawfulness
PS C:\Users\Luke Pepin\OneDrive\Documents\Academics\PA1> python .\multi_thread_client.py
Enter your wildcard query (e.g., 'a*'): ha?y
Sending query: ha?y
Received response:
200 OK
Number of matching words: 43
cheerful
chalky
cherry
cheeky
chatty

```

Figure 9: Multi-Thread Success Multiple Wildcards