Socket Programming

Abdelrahman Wael 20010869, Ahmed Hesham 20010234

Introduction

This socket programming assignment explores the development of a robust server-client architecture, implementing key features such as handling HTTP requests, supporting persistent connections, and managing concurrent clients. The focus is on creating a system capable of processing various HTTP methods, enabling communication between a web server and clients.

Assumptions

- **Multithreading VS Multiprocessing**: We decided to implement the connections using a multithreaded approach as it seems more fit for a client-server architecture:
    - **Reducing Inter-process communication (IPC)** as threads share the same memory space, thus resource efficient
    - **Less overhead** as process creation can be almost 10x more costly than thread creation
    - **Speed and responsiveness** as threads communicate faster together
- **Timeout:** For any connection there's a 10 second timeout. This time decreases when number of active connections increases (10 / number of active connections)
- **New connections:** When a new connection is made it is handled by a new thread, this process is made by the ***"master socket"*** where its responsible for accepting new connections and assigning a thread for each
- **Queue**: At most 3 connections waiting in the backlog to be accepted. This assumption mainly depends on the available memory resources as they vary from each server.

How to Run

1. Open server directory and compile file (gcc server.c -o server –lpthread)
2. Run server on any available port i.e. 8080 (./server 8080)
3. Open client directory and compile file (gcc client.c -o client)
4. Run client (/.client 127.0.0.1 8080)

Implementation

Server

- Listening for connections using master_socket

```
if (listen(master_socket, 3) < 0)
{
perror("listen");
exit(EXIT_FAILURE);
}
```

- Accepting new connection and delegation to new thread

```
createMasterSocket();
while (1)
{
int new_socket = acceptConnection();

//this is atomic to count the number of sockets correctly
pthread_mutex_lock(&lock);
FD_SET(new_socket, &fds);
counter++;
pthread_mutex_unlock(&lock);
//a new thread will handle new connections
pthread_t thread_id;
int *pclient = malloc(sizeof(int));
*pclient = new_socket;
pthread_create(&thread_id, NULL, handle_connection, pclient);
}
```

- Handling GET and POST request

```
void handleGetRequest(char *path, int sd){
sendFile(path, sd);
}

void handlePostRequest(char *path, int sd){
sendMessage(sd, ok_msg);
receiveFile(path, sd);
}
```

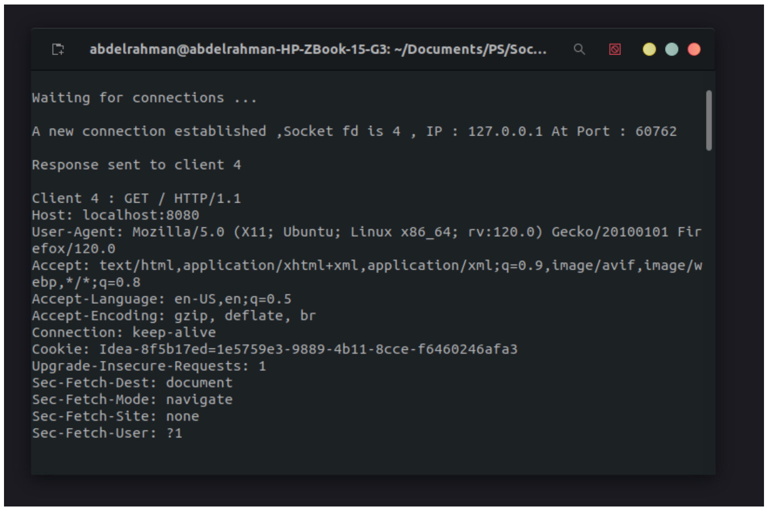
- Persistent connection, Dynamic timeout , Handling connection

```
pthread_mutex_lock(&lock);
timeout.tv_sec = 10/counter; //congestion dependent
pthread_mutex_unlock(&lock);
```

```
while (1)
{
int t = select(FD_SETSIZE, &fds, NULL, NULL, &timeout);
if (t == 0){
printf("\nTimeout\n");
closeConnection(sd);
return 0;
}
read_size = read(sd, client_message, MAX_BUFFER_SIZE);
if (read_size == -1){
perror("recv failed");
}
printf("\nClient %d : %s \n", sd, client_message);
if (isCloseMessage(client_message)){
closeConnection(sd);
break;
}
parseMessage(client_message, type, path);
if (strcmp(type, "POST") == 0){
handlePostRequest(path, sd);
}
else if (strcmp(type, "GET") == 0){
handleGetRequest(path, sd);
}
memset(client_message, 0, MAX_BUFFER_SIZE);memset(type, 0, 256);memset(path, 0, 256);}
```

Bonus

Testing in a real browser



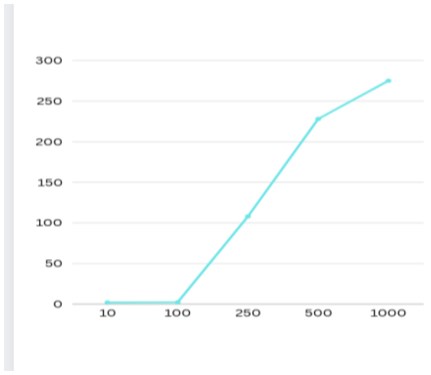
abdelrahman@abdelrahman-HP-ZBook-15-G3: ~/Documents/PS/Soc...

```
Waiting for connections ...

A new connection established ,Socket fd is 4 , IP : 127.0.0.1 At Port : 60762

Response sent to client 4

Client 4 : GET / HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:120.0) Gecko/20100101 Fir
efox/120.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Cookie: Idea-8f5b17ed=1e5759e3-9889-4b11-8cce-f6460246afa3
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
```

Performance Evaluation

Using Apache Bench: ab -n x (i.e. 1000) -c 10 http://127.0.0.1:8080/

Test #	Total requests	Time delay per request (ms)
1	10	2.32
2	100	2.3
3	250	108
4	500	228.154
5	1000	274.98



300
250
200
150
100
50
0
10
100
250
500
1000