NETWORK PROGRAMMING REPORT
Comparison of Iterative and Concurrent Servers

1. Introduction
In this report, we compare iterative and concurrent servers, analyzing their advantages,
disadvantages, real-world applications, and system resource usage.

2. Iterative Server
- Handles one client at a time.
- Processes the client request, then closes the connection before accepting another.
- Simple to implement but slow for handling multiple clients.

Advantages:
- Low resource usage.
- Easy to develop and debug.

Disadvantages:
- Poor scalability (blocks new clients until the current one finishes).
- Not suitable for real-time applications.

Real-World Applications:
- FTP servers (single-user mode).
- Debugging or testing environments.

3. Concurrent Server
- Can handle multiple clients at the same time using threads or processes.
- More efficient for large-scale applications.

Advantages:
- Fast response time.
- Can serve many clients simultaneously.
- Ideal for high-traffic applications.

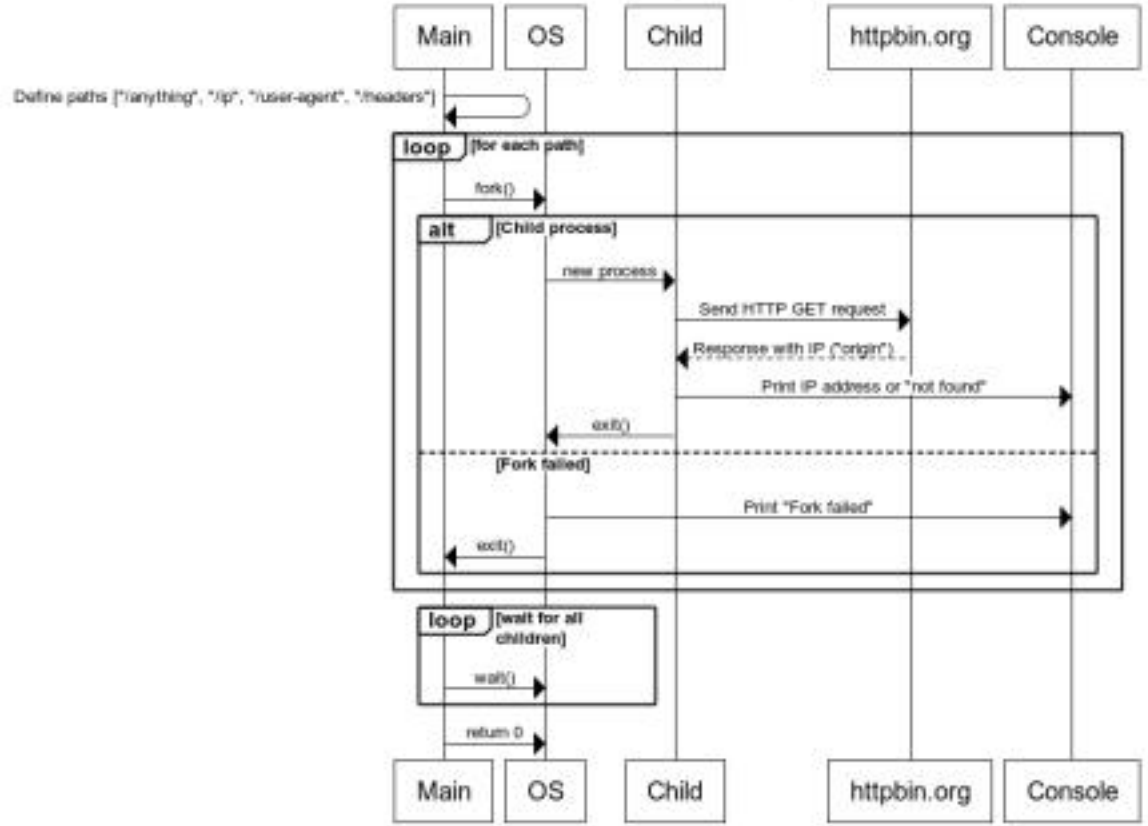
Disadvantages:
- Higher resource consumption.
- More complex to implement due to thread synchronization issues.

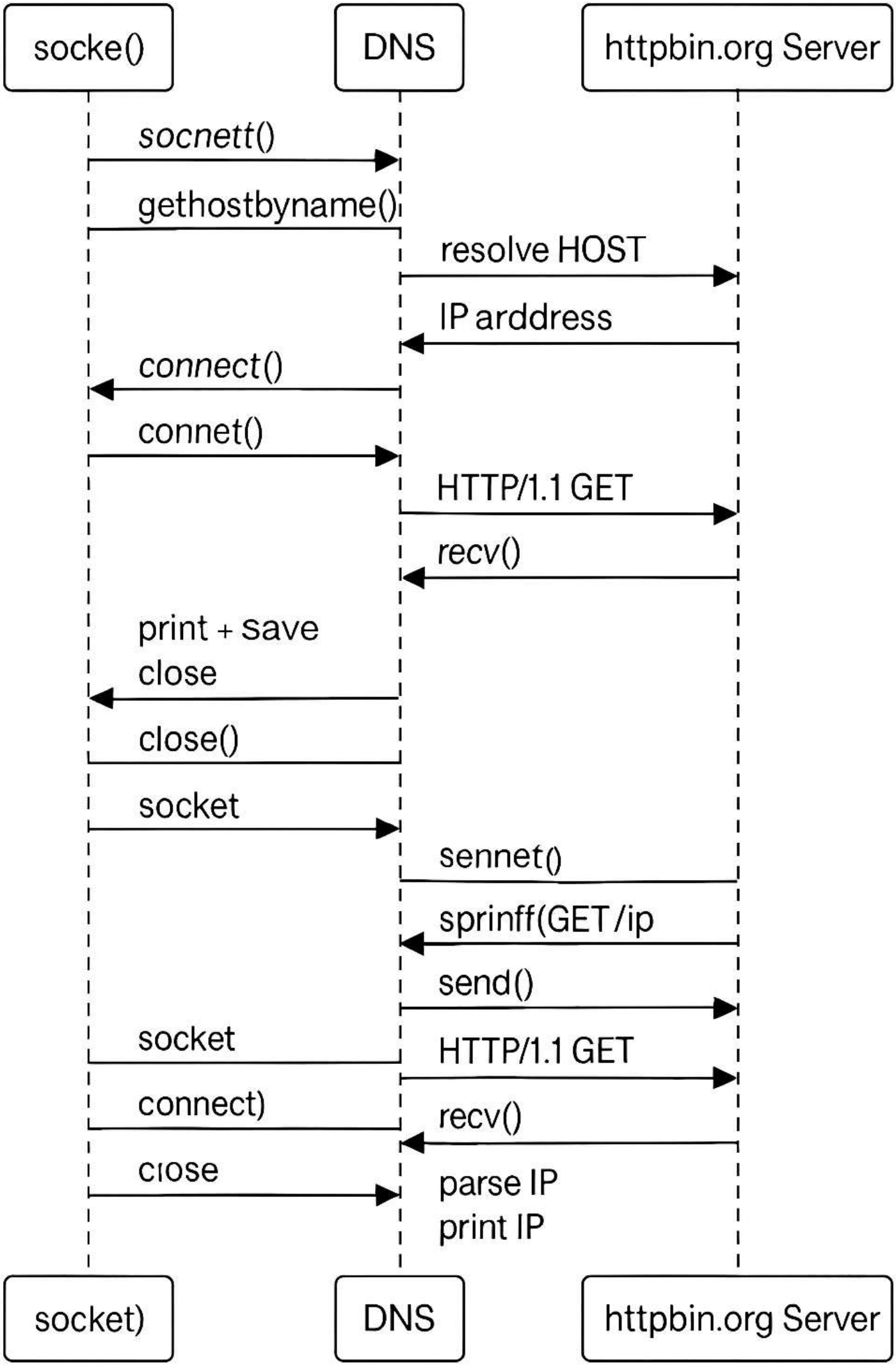
Real-World Applications:
- Web servers (Apache, Nginx).
- Online multiplayer games.
- Real-time stock market systems.

4. System Resource Usage Analysis
- Iterative servers use minimal resources but are slow under heavy load.
- Concurrent servers handle high loads but consume more CPU and memory.
- System resource usage can be monitored using the command:
  `ps aux | grep server`

5. Conclusion
- Iterative servers are useful for small applications but lack scalability.
- Concurrent servers are preferred for modern applications requiring multiple connections.

Concurrent HTTP Client (Simplified Logic)




socke()
DNS
httpbin.org Server
socnett()
gethostbyname()
resolve HOST
IP arddress
connect()
connet()
HTTP/1.1 GET
recv()
print + save
close
close()
socket
sennet()
sprinff(GET /ip
send()
socket
HTTP/1.1 GET
connect)
recv()
close
parse IP
print IP
socket)
DNS
httpbin.org Server


1 2 3 4
17:39
mirza@kali: ~/Desktop
File Actions Edit View Help
Saving /stream/1 response to stream.html ...
HTTP/1.1 200 OK
Date: Tue, 01 Apr 2025 14:16:43 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Connection: close
Server: gunicorn/19.9.0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
e8
{"url": "http://httpbin.org/stream/1", "args": {}, "headers": {"Host": "httpbin.org", "X-Amzn-Trace-Id": "Root=1-67ebf54b-52df3215462c63ae652c75e0", "User-Agent": "Mozilla/5.0", "Accept": "*/*"}, "origin": "90.131.46.154", "id": 0}
0
Fetching client IP address ...
Client IP: 90.131.46.154
┌──(mirza㉿kali)-[~/Desktop]
└─$ gcc Tcp-client.c -o tcp-client
┌──(mirza㉿kali)-[~/Desktop]
└─$ ./tcp-client
Saving /stream/1 response to stream.html ...
HTTP/1.1 200 OK
Date: Tue, 01 Apr 2025 14:24:10 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Connection: close
Server: gunicorn/19.9.0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
e8
{"url": "http://httpbin.org/stream/1", "args": {}, "headers": {"Host": "httpbin.org", "X-Amzn-Trace-Id": "Root=1-67ebf70a-43b9d0e80f0588d34e35e2d5", "User-Agent": "Mozilla/5.0", "Accept": "*/*"}, "origin": "90.131.46.154", "id": 0}
0
Fetching client IP address ...
Client IP: 90.131.46.154
┌──(mirza㉿kali)-[~/Desktop]
└─$


mirza@kali: ~/Desktop
File Actions Edit View Help
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
{
"user-agent": "Mozilla/5.0"
}
── End of /user-agent ──
HTTP/1.1 200 OK
Date: Tue, 01 Apr 2025 14:25:23 GMT
Content-Type: application/json
Content-Length: 32
Connection: close
Server: gunicorn/19.9.0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
{
"origin": "90.131.46.154"
}
── End of /ip ──
(mirza㉿kali)-[~/Desktop]
$
(mirza㉿kali)-[~/Desktop]
$ gcc concurrentt-client.c -o concurrentt-client
(mirza㉿kali)-[~/Desktop]
$ ./concurrentt-client
/anything → IP address found: 90.131.46.154
/ip → IP address found: 90.131.46.154
/user-agent → IP address not found
/headers → IP address not found
(mirza㉿kali)-[~/Desktop]
$ gcc concurrentt-client.c -o concurrentt-client
(mirza㉿kali)-[~/Desktop]
$ ./concurrentt-client
/ip → IP address found: 90.131.46.154
/anything → IP address found: 90.131.46.154
/user-agent → IP address not found
/headers → IP address not found
(mirza㉿kali)-[~/Desktop]
$
Screenshot taken
View image

Source Code: Tcp-client.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netdb.h>
#include <arpa/inet.h>

#define PORT 80
#define HOST "httpbin.org"

int main() {
    int sock;
    struct sockaddr_in server_addr;
    struct hostent *server;
    char request[1024], response[4096];
    int bytes_received;

    // ----------------------------
    // Step 1: GET /stream/1
    // ----------------------------

    // Create socket
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        perror("Socket error");
        return 1;
    }

    // Get host info
    server = gethostbyname(HOST);
    if (server == NULL) {
        fprintf(stderr, "No such host\n");
        return 1;
    }

    // Set up the server address
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    memcpy(&server_addr.sin_addr.s_addr, server->h_addr, server->h_length);

    // Connect to the server
    if (connect(sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Connection failed");
        return 1;
    }

    // Send HTTP GET for /stream/1 with headers to avoid 502 error
    sprintf(request,
        "GET /stream/1 HTTP/1.1\r\n"
        "Host: %s\r\n"
```

```
        "User-Agent: Mozilla/5.0\r\n"
        "Accept: */*\r\n"
        "Connection: close\r\n\r\n", HOST);
    send(sock, request, strlen(request), 0);

    // Save response to file
    FILE *file = fopen("stream.html", "w");
    if (!file) {
        perror("File error");
        return 1;
    }

    printf("Saving /stream/1 response to stream.html...\n");
    while ((bytes_received = recv(sock, response, sizeof(response) - 1, 0)) > 0) {
        response[bytes_received] = '\0';
        printf("%s", response);       // Print to terminal
        fprintf(file, "%s", response); // Save to file
    }

    fclose(file);
    close(sock);

    // ----------------------------
    // Step 2: GET /ip and extract IP
    // ----------------------------

    // Create new socket
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        perror("Socket error (IP)");
        return 1;
    }

    // Reconnect to the server
    if (connect(sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Connection failed (IP)");
        return 1;
    }

    // Send HTTP GET for /ip
    sprintf(request, "GET /ip HTTP/1.1\r\nHost: %s\r\nConnection: close\r\n\r\n", HOST);
    send(sock, request, strlen(request), 0);

    printf("\nFetching client IP address...\n");

    // Receive full response into a buffer
    char full_response[8192] = "";
    while ((bytes_received = recv(sock, response, sizeof(response) - 1, 0)) > 0) {
        response[bytes_received] = '\0';
        strcat(full_response, response);
    }

    // Extract the IP address
    char *origin = strstr(full_response, "\"origin\":");
```

```
    if (origin != NULL) {
        char ip[64];
        if (sscanf(origin, "\"origin\": \"%63[^\"]\"", ip) == 1) {
            printf("\nClient IP: %s\n", ip);
        } else {
            printf("\nCould not parse IP address.\n");
        }
    } else {
        printf("\n'origin' field not found in response.\n");
    }

    close(sock);
    return 0;
}
```

Source Code: concurrentt-client.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/wait.h>

#define PORT 80
#define HOST "httpbin.org"

void fetch_ip(const char *path) {
    int sock;
    struct sockaddr_in server_addr;
    struct hostent *server;
    char request[1024], response[4096], full_response[8192] = "";
    int bytes_received;

    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) exit(1);

    server = gethostbyname(HOST);
    if (!server) exit(1);

    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    memcpy(&server_addr.sin_addr.s_addr, server->h_addr, server->h_length);

    if (connect(sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) exit(1);

    sprintf(request,
        "GET %s HTTP/1.1\r\n"
        "Host: %s\r\n"
        "User-Agent: Mozilla/5.0\r\n"
        "Connection: close\r\n\r\n", path, HOST);
    send(sock, request, strlen(request), 0);

    while ((bytes_received = recv(sock, response, sizeof(response) - 1, 0)) > 0) {
        response[bytes_received] = '\0';
        strcat(full_response, response);
    }

    // Try to find and extract IP
    char *origin = strstr(full_response, "\"origin\":");
    if (origin) {
        char ip[64];
        if (sscanf(origin, "\"origin\": \"%63[^\"]\"", ip) == 1) {
            printf("%s -> IP address found: %s\n", path, ip);
        } else {
            printf("%s -> IP address not found\n", path);
```

```
        }
    } else {
        printf("%s -> IP address not found\n", path);
    }

    close(sock);
}

int main() {
    const char *paths[] = { "/anything", "/ip", "/user-agent", "/headers" };
    int num_paths = sizeof(paths) / sizeof(paths[0]);

    for (int i = 0; i < num_paths; i++) {
        pid_t pid = fork();

        if (pid == 0) {
            fetch_ip(paths[i]);
            exit(0);
        } else if (pid < 0) {
            perror("Fork failed");
            exit(1);
        }
    }

    for (int i = 0; i < num_paths; i++) {
        wait(NULL);
    }

    return 0;
}
```