Sequence of Actions for HTTP Server with Sockets and Epoll

1 Introduction

This document explains the sequence of actions required to build an HTTP server using sockets, epoll, and the HTTP protocol in C++.

2 1. Creating the Server Socket

The server socket listens for incoming client connections.

Steps:

- Create a socket using socket(AF_INET, SOCK_STREAM, 0).
- Bind the socket to an IP address and port using bind().
- Start listening with listen().

Listing 1: Creating a Server Socket

```
int server_fd = socket(AF_INET, SOCK_STREAM, 0);
sockaddr_in server_addr = {AF_INET, htons(8080), INADDR_ANY};
bind(server_fd, (sockaddr*)&server_addr, sizeof(server_addr));
listen(server_fd, SOMAXCONN);
```

3 2. Initializing Epoll

Epoll is used to monitor multiple sockets efficiently.

Steps:

- Create an epoll instance with epoll_create1(0).
- Add the server socket to epoll using epoll_ctl().
- Use epoll_wait() to monitor events.

Listing 2: Initializing Epoll

```
int epoll_fd = epoll_create1(0);
epoll_event event;
event.events = EPOLLIN;
event.data.fd = server_fd;
epoll_ctl(epoll_fd, EPOLL_CTL_ADD, server_fd, &event);
```

4 3. Main Loop: Handling Connections

Listing 3: Handling New Connections

```
epoll_event events[10];
while (true) {
    int num_events = epoll_wait(epoll_fd, events, 10, -1);
    for (int i = 0; i < num_events; i++) {
        if (events[i].data.fd == server_fd) {
            int client_fd = accept(server_fd, nullptr, nullptr);
            epoll_event client_event;
            client_event.events = EPOLLIN;
            client_event.data.fd = client_fd;
            epoll_ctl(epoll_fd, EPOLL_CTL_ADD, client_fd, &
                client_event);
        }
    }
}
```

5 4. Reading Client Data

Listing 4: Reading HTTP Requests

```
if (events[i].events & EPOLLIN) {
    char buffer[4096];
    int bytes_read = read(events[i].data.fd, buffer, sizeof(buffer)
        );
    if (bytes_read <= 0) {
        close(events[i].data.fd);
    } else {
        // Process the HTTP request
    }
}
```

6 5. Parsing the HTTP Request

Extract method, URI, and version.

Listing 5: Parsing HTTP Request

```
std::string request(buffer);
std::istringstream request_stream(request);
```

```cpp
std::string method, uri, version;
request_stream >> method >> uri >> version;
```

7 6. Sending an HTTP Response

Listing 6: Generating HTTP Response

```cpp
std::string response = "HTTP/1.1 200 OK\r\n"
                       "Content-Type: text/plain\r\n"
                       "Content-Length: 13\r\n"
                       "\r\n"
                       "Hello, World!";
write(events[i].data.fd, response.c_str(), response.size());
close(events[i].data.fd);
```

8 7. Complete Server Code

Listing 7: Full HTTP Server with Epoll

```cpp
#include <iostream>
#include <sys/epoll.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <cstring>
#include <sstream>

constexpr int MAX_EVENTS = 10;
constexpr int PORT = 8080;

int main() {
    int server_fd = socket(AF_INET, SOCK_STREAM, 0);
    sockaddr_in server_addr = {AF_INET, htons(PORT), INADDR_ANY};
    bind(server_fd, (sockaddr*)&server_addr, sizeof(server_addr));
    listen(server_fd, SOMAXCONN);

    int epoll_fd = epoll_create1(0);
    epoll_event event, events[MAX_EVENTS];
    event.events = EPOLLIN;
    event.data.fd = server_fd;
    epoll_ctl(epoll_fd, EPOLL_CTL_ADD, server_fd, &event);

    while (true) {
        int num_events = epoll_wait(epoll_fd, events, MAX_EVENTS,
            -1);
        for (int i = 0; i < num_events; ++i) {
            if (events[i].data.fd == server_fd) {
                int client_fd = accept(server_fd, nullptr, nullptr)
                    ;
                event.events = EPOLLIN;
                event.data.fd = client_fd;
                epoll_ctl(epoll_fd, EPOLL_CTL_ADD, client_fd, &event);
            } else {
                char buffer[4096];
                int bytes_read = read(events[i].data.fd, buffer, sizeof(buffer));
                if (bytes_read <= 0) {
                    close(events[i].data.fd);
                } else {
                    std::string request(buffer);
                    std::istringstream request_stream(request);
                    std::string method, uri, version;
                    request_stream >> method >> uri >> version;

                    std::string response = "HTTP/1.1 200 OK\r\n"
                                           "Content-Type: text/plain\r\n"
                                           "Content-Length: 13\r\n"
                                           "\r\n"
                                           "Hello, World!";
                    write(events[i].data.fd, response.c_str(), response.size());
                    close(events[i].data.fd);
                }
            }
        }
    }
    close(server_fd);
    close(epoll_fd);
    return 0;
}
```