This C code is a multi-threaded HTTP server designed to handle requests for HTML and JPEG files. The server listens on a specified port, accepts incoming connections, and serves the requested files to clients. If the server is handling more than 10 requests simultaneously, it will reject additional requests until some threads complete. Below is a detailed breakdown of the code:

Header Inclusions and Macros

c
Copy code
```
#include <stdio.h>
#include <string.h>    // for strlen
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h> // for inet_addr
#include <unistd.h>    // for write
#include <pthread.h>   // for threading, link with lpthread
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <semaphore.h>

#define PATH "/home/ozgur/workspace/assignment2/sources" // Path to
the directory containing files
#define PORT_NO 8888
#define BUFFER_SIZE 1024
#define CONNECTION_NUMBER 10
```

- **Standard Libraries**: Included for standard I/O, string manipulation, memory allocation, etc.
- **Socket Libraries**: Included for creating and managing sockets.
- **Threading Library**: Included for creating threads.
- **Semaphore Library**: Included for managing access control using semaphores.
- **Macros**: Defines constants for the file path, port number, buffer size, and maximum number of simultaneous connections.

Global Variables and Semaphore

c
Copy code
```
int thread_count = 0; // Keeps track of the number of active threads.
```

```
sem_t mutex; // Semaphore to control access to the thread_count
variable.
```

- `thread_count`: Counter for the number of active threads.
- `mutex`: Semaphore to manage concurrent access to `thread_count`.

Function to Handle JPEG Files

c
Copy code

```
void jpeg_handler(int socket, char *file_name) {
    char *buffer;
    char *full_path = (char *)malloc((strlen(PATH) +
strlen(file_name)) * sizeof(char));
    int fp;

    strcpy(full_path, PATH);
    strcat(full_path, file_name);
    puts(full_path);

    if ((fp = open(full_path, O_RDONLY)) > 0) {
        puts("Image Found.");
        int bytes;
        char buffer[BUFFER_SIZE];

        send(socket, "HTTP/1.0 200 OK\r\nContent-Type:
image/jpeg\r\n\r\n", 45, 0);
        while ((bytes = read(fp, buffer, BUFFER_SIZE)) > 0)
            write(socket, buffer, bytes);
    } else {
        write(socket, "HTTP/1.0 404 Not Found\r\nConnection:
close\r\nContent-Type: text/html\r\n\r\n<!doctype html><html><body>404
File Not Found</body></html>", strlen("HTTP/1.0 404 Not
Found\r\nConnection: close\r\nContent-Type: text/html\r\n\r\n<!doctype
html><html><body>404 File Not Found</body></html>"));
    }

    free(full_path);
    close(fp);
}
```

- **`jpeg_handler`**: Handles requests for JPEG files.
    - Constructs the full file path.
    - Checks if the file exists and sends the file in chunks.
    - Sends a 404 response if the file is not found.

Function to Handle HTML Files

c
Copy code

```c
void html_handler(int socket, char *file_name) {
    char *buffer;
    char *full_path = (char *)malloc((strlen(PATH) +
strlen(file_name)) * sizeof(char));
    FILE *fp;

    strcpy(full_path, PATH);
    strcat(full_path, file_name);

    fp = fopen(full_path, "r");
    if (fp != NULL) {
        puts("File Found.");

        fseek(fp, 0, SEEK_END);
        long bytes_read = ftell(fp);
        fseek(fp, 0, SEEK_SET);

        send(socket, "HTTP/1.0 200 OK\r\nContent-Type:
text/html\r\n\r\n", 44, 0);
        buffer = (char *)malloc(bytes_read * sizeof(char));

        fread(buffer, bytes_read, 1, fp);
        write(socket, buffer, bytes_read);
        free(buffer);

        fclose(fp);
    } else {
        write(socket, "HTTP/1.0 404 Not Found\r\nConnection: 
close\r\nContent-Type: text/html\r\n\r\n<!doctype html><html><body>404 
File Not Found</body></html>", strlen("HTTP/1.0 404 Not 
Found\r\nConnection: close\r\nContent-Type: text/html\r\n\r\n<!doctype 
html><html><body>404 File Not Found</body></html>"));
    }

    free(full_path);
}
```

- **`html_handler`**: Handles requests for HTML files.
  - Constructs the full file path.
  - Checks if the file exists and sends the entire file.
  - Sends a 404 response if the file is not found.

Connection Handler Function

c
Copy code

```
void *connection_handler(void *socket_desc) {
    int request;
    char client_reply[BUFFER_SIZE], *request_lines[3];
    char *file_name;
    char *extension;

    int sock = *((int *)socket_desc);
    request = recv(sock, client_reply, BUFFER_SIZE, 0);

    sem_wait(&mutex);
    thread_count++;

    if (thread_count > 10) {
        char *message = "HTTP/1.0 400 Bad Request\r\nContent-Type: 
text/html\r\n\r\n<!doctype html><html><body>System is busy right 
now.</body></html>";
        write(sock, message, strlen(message));
        thread_count--;
        sem_post(&mutex);
        free(socket_desc);
        shutdown(sock, SHUT_RDWR);
        close(sock);
        pthread_exit(NULL);
    }
    sem_post(&mutex);

    if (request < 0) {
        puts("Recv failed");
    } else if (request == 0) {
        puts("Client disconnected unexpectedly.");
    } else {
        printf("%s", client_reply);
        request_lines[0] = strtok(client_reply, " \t\n");
        if (strncmp(request_lines[0], "GET\0", 4) == 0) {
            request_lines[1] = strtok(NULL, " \t");
            request_lines[2] = strtok(NULL, " \t\n");

            if (strncmp(request_lines[2], "HTTP/1.0", 8) != 0 &&
strncmp(request_lines[2], "HTTP/1.1", 8) != 0) {
                char *message = "HTTP/1.0 400 Bad
Request\r\nConnection: close\r\nContent-Type:
text/html\r\n\r\n<!doctype html><html><body>400 Bad
Request</body></html>";
                write(sock, message, strlen(message));
            } else {
                char *tokens[2];

                file_name = (char *)malloc(strlen(request_lines[1]) *
sizeof(char));
                strcpy(file_name, request_lines[1]);
                puts(file_name);

                tokens[0] = strtok(file_name, ".");
                tokens[1] = strtok(NULL, ".");

                if (strcmp(tokens[0], "/favicon") == 0 &&
strcmp(tokens[1], "ico") == 0) {
                    sem_wait(&mutex);
                    thread_count--;
                    sem_post(&mutex);
                    free(socket_desc);
                    shutdown(sock, SHUT_RDWR);
                    close(sock);
                    pthread_exit(NULL);
                } else if (tokens[0] == NULL || tokens[1] == NULL) {
                    char *message = "HTTP/1.0 400 Bad
Request\r\nConnection: close\r\n\r\n<!doctype html><html><body>400 Bad
Request. (You need to request to jpeg and html files)</body></html>";
                    write(sock, message, strlen(message));
                } else {
                    if (strcmp(tokens[1], "html") != 0 &&
strcmp(tokens[1], "jpeg") != 0) {
                        char *message = "HTTP/1.0 400 Bad
Request\r\nConnection: close\r\n\r\n<!doctype html><html><body>400 Bad
Request. Not Supported File Type (Supported File Types: html and
jpeg)</body></html>";
                        write(sock, message, strlen(message));
                    } else {
                        if (strcmp(tokens[1], "html") == 0) {
                            sem_wait(&mutex);
                            html_handler(sock, request_lines[1]);
                            sem_post(&mutex);
                        } else if (strcmp(tokens[1], "jpeg") == 0) {
                            sem_wait(&mutex);
                            jpeg_handler(sock, request_lines[1]);
                            sem_post(&mutex);
                        }
                    }
                }
                free(file_name);
            }
        }
    }

    free(socket_desc);
    shutdown(sock, SHUT_RDWR);
    close(sock);
    sem_wait(&mutex);
    thread_count--;
    sem_post(&mutex);
    pthread_exit(NULL);
}
```

- **`connection_handler`**: Handles each client connection.
  - Receives the HTTP request.
  - Increments `thread_count` and checks if the server is too busy.
  - Parses the request to determine if it is a `GET` request.
  - Handles file requests based on their extension (HTML or JPEG).
  - Sends appropriate responses for invalid requests or file not found.
  - Decrements `thread_count` after handling the request.

Main Function

c
Copy code

```
int main(int argc, char *argv[]) {
    sem_init(&mutex, 0, 1);
    int socket_desc, new_socket, c, *new_sock;
    struct sockaddr_in server, client;

    socket_desc = socket(AF_INET, SOCK_STREAM, 0);
    if (socket_desc == -1) {
        puts("Could not create socket");
        return 1;
    }

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(PORT_NO);

    if (bind(socket_desc, (struct sockaddr *)&server, sizeof(server))
< 0) {
        puts("Binding failed");
        return 1;
    }
    listen(socket_desc, 20);

    puts("Waiting for incoming connections...");
    c = sizeof(struct sockaddr_in);
    while ((new_socket = accept(socket_desc, (struct sockaddr
*)&client, (socklen_t *)&c))) {
        puts("Connection accepted \n");

        pthread_t sniffer_thread;
        new_sock = malloc(1);
        *new_sock = new_socket;

        if (pthread_create(&sniffer_thread, NULL, connection_handler,
(void *)new_sock) < 0) {
            puts("Could not create thread");
            return 1;
        }
    }

    return 0;
}
```

- **`main`**: Sets up the server and handles incoming connections.
    - Initializes the semaphore.
    - Creates the server socket.
    - Binds the socket to the specified port.
    - Listens for incoming connections.
    - Accepts connections and creates a new thread for each connection.

This server code is a simple yet effective way to handle HTTP requests for specific file types (HTML and JPEG) while managing concurrency using threads and semaphores.

4o