COMP 4981
Assignment 1

server.c server

Design

Wilson Sue & Roy Xavier Pimentel
A01266055 & A00697839
Feb 4th, 2024

States

States	Description
socket_create	Creates a domain socket using socket() system call.
socket_bind	Server binds the domain socket to an IP address and port using bind().
socket_listen	Server listens for incoming connections on the bound socket using listen().
accept_connection	Server accepts an incoming connection using accept(), establishing a new socket for the client.
initialize_server_socket	Initial setup of the server, including starting the database, setting up the listening socket, etc.
start_server	Main loop of the server, where it waits for and responds to connections and requests.
select	Call to make the array of file descriptors for IO multiplexing
read_request	Reads data from the client socket into a buffer.
parse_request	Interprets the HTTP request received from the client to determine its type (GET, POST, HEAD, etc.).
HEAD_request	Handles a HEAD request, returning the headers for the requested resource without the body.
HEAD_Response	Constructs and sends the headers for a HEAD response without the body content.
GET_Request	Identifies and handles a GET request, fetching the requested file or resource.
open_file	Reads any file from /webroot/ but defaults to index.html if / detected also
404_response	If file not found on webroot/ default to 404.html
GET_response	Prepares the HTTP response for a GET request, including headers and the requested file content.
POST_request	Processes a POST request typically involves receiving and handling data sent in the request body.
open_database	Opens database when POST request received
store_stringInDb	POST data gets attached a generic key and overwrites any previous data inside database

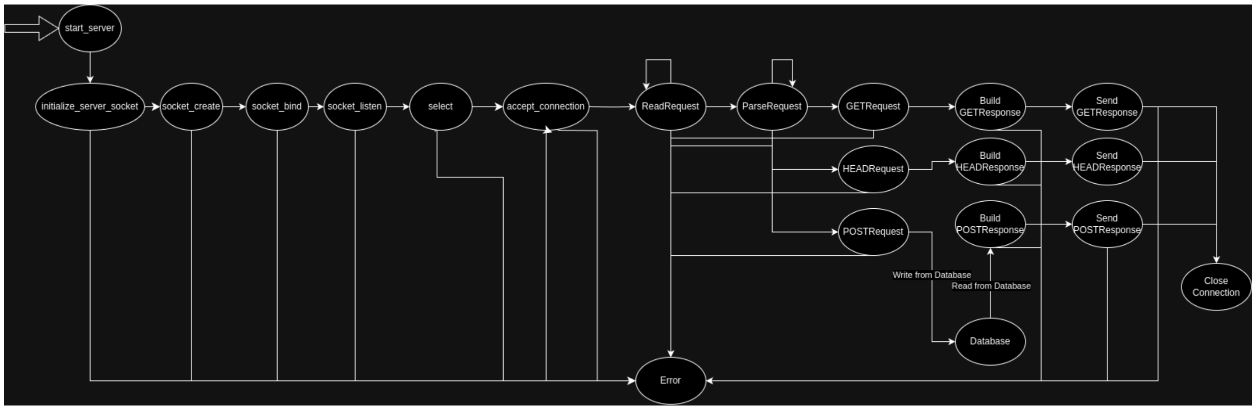
read_stringFromDb	
POST_response	Assembles the response for a POST request, possibly including confirmation of data receipt or processing results.
close_connection	Closes the client socket after the request has been processed and the response sent.
Error	Handles error conditions, such as request parsing failures, file not found errors, and internal server errors.
Exit	Exit Gracefully

State Table

From State	To State	Condition | Description | Function Call
server server	server initialization	initialize_server_socket()
server initialization	socket bind	bind()
socket bind	socket listen	listen()
socket listen	select	IO multiplexing for accept_connection()
select	accept connection	Assigns file descriptor for each connection
socket_accept	read request	Reading data from socket
read_request	parse request	Parsing HTTP request
parse_request	GET request	If HTTP GET request detected
parse_request	HEAD request	If HTTP HEAD request detected
parse_request	POST request	If HTTP POST request detected

GET_request	open file	Construct response for GET request Then sends it
open_file	GET response	send() file response to client
open_file	404 response	
HEAD_request	HEAD response	Construct response for HEAD request Then sends it
POST_request	open_database	Construct response for POST request
open_database	store_stringInDb	POST data gets attached a generic key and overwrites any previous
store_stringInDb	read_stringFromDb	String gets read from db and attached to a response
read_stringFromDb	POST response	Read data added to response and sent
GET_response	close connection	close() socket
404_response	close connection	close() socket
HEAD_response	close connection	close() socket
POST_response	close connection	close() socket
Any state	Error Handling	On encountering errors
Error Handling	close connection	close() socket on error

State Transition Diagram



Functions

Function	Description
initialize_server_socket()	Initializes the server socket, sets socket options, and returns the socket descriptor.
bind()	Binds the server socket to a specific IP address and port number.
listen()	Puts the server socket in a listening state to accept incoming connections.
accept_connection()	Accepts a new connection from a client and returns a new socket descriptor for this connection.
openDatabase()	Opens a connection to the database file specified by dbName.
storeStringInDB()	Stores a given string in the database.
readStringFromDB()	Reads a string from the database.
closeDatabase()	Closes the open database connection.

send()	Sends data over a socket connection.
close()	Closes a socket or file descriptor, terminating the connection.
read()	Reads data from a socket into a buffer.
printf()	Outputs formatted data to standard output.
perror()	Prints a descriptive error message to stderr based on the value of the global errno.
exit()	Terminates the program execution with an exit status.
socket()	Creates an endpoint for communication and returns a socket descriptor.
setsockopt()	Sets options on the socket.
inet_addr()	Converts the Internet host address from IPv4 numbers-and-dots notation into binary data.
htons()	Converts the unsigned short integer hostshort from host byte order to network byte order.
FD_ZERO()	Clears a set.
FD_SET()	Adds a descriptor to a set.
select()	Monitors multiple file descriptors to see if any of them is ready for reading, writing, or if there is an exceptional condition pending.
strstr()	Finds the first occurrence of a substring in a string.
sscanf()	Reads formatted input from a string.

malloc()	Allocates a block of memory on the heap.
strncpy()	Copies a specified number of characters from one string to another.
free()	Frees allocated memory.
stat()	Retrieves information about the file pointed to by path.

States

parse_arguments

Parameters

Parameter	Description
argc	The number of command line arguments passed to main
argv	The command line arguments passed to main
listen_ip	The ip address to listen on
listen_port	The port to listen on
proxy_ip	The ip address to send data to
proxy_port	The port to send data to

Return

- Nothing

Pseudo Code

```
call getopt in a loop
     if opt is ‘h’
          call usage
     if opt is ‘i’
          set listen_ip to argv[optind]
     if opt is ‘p’
          set listen_port to argv[optind]
     if opt is ‘I’
          set proxy_ip to argv[optind]
     if opt is ‘P’
          set proxy_port to argv[optind]
     If opt is ‘?’
          call usage

If there were more than 4 arguments
     call usage
```

parse_arguments

Parameters

Parameter	Description
binary_name	The name of the executable
listen_ip	The ip address to listen on
listen_port	The port to listen on
proxy_ip	The ip address to send data to
proxy_port	The port to send data to
settings	Settings to hold the converted parameters

Return

- nothing

Pseudo Code

```
if listen_ip_str is not set
     call usage
if listen_port_str is not set
     call usage
if proxy_ip_str is not set
     call usage
if proxy_port_str is not set
     call usage

call
```

create_socket

Description

Creates a new socket and initializes it for server communication. The socket is configured with the provided IP address and port number.

Parameters

Parameter	Description
address	IP address to bind to
port	Port number to bind to

Return

Integer representing the socket file descriptor

Pseudo Code

```
if listen_ip_str is not set
    call usage
if listen_port_str is not set
    call usage
if proxy_ip_str is not set
    call usage
if proxy_port_str is not set
    call usage
```

socket_bind

Description

The bind function associates the server socket with a specific IP address and port.

Parameters

Parameter	Description
server_socket	The file descriptor representing the server socket.
server_addr	A pointer to a structure containing the server address information.

Return

-If the bind operation succeeds, it returns 0.
-If an error occurs during the bind operation, it returns -1, and an error message can be obtained using the errno variable or by calling perror.

Pseudo Code

```
if listen_ip_str is not set
    call usage
if listen_port_str is not set
    call usage
if proxy_ip_str is not set
    call usage
if proxy_port_str is not set
    call usage
```

socket_listen

Description

The listen function sets the server socket to listen for incoming connections from clients.

Parameters

Parameter	Description
server socket	An integer representing the server socket descriptor.
MAX_CLIENTS	An integer specifying the maximum number of pending connections that can be queued for the server socket

Return

This function does not return a value. It either succeeds or fails, terminating the program if it fails.

Pseudo Code

```
Function listen_server_socket(server_socket, max_clients):
    Attempt to start listening on the server socket for incoming
connections with a maximum queue size of max_clients.
    If the attempt fails:
        Print an error message indicating that listening failed.
        Close the server socket.
        Exit the program with a failure status.
```

start_server

Parameters

Parameter	Description
address	The IP address on which the server will listen for incoming connections.
port	The port number on which the server will listen for incoming connections.

Return

none

Pseudo Code

```
Function start_server(address, port):
    Initialize server socket
    Print server listening message

    While true:
        Wait for incoming connections
        Accept incoming connection
        Handle incoming data from clients
```

select

Parameters

Parameter	Description
max_sd	The highest-numbered file descriptor in any of the three sets, plus 1.
read_fd	A pointer to a set of file descriptors to be monitored for readability.
write_fd	A pointer to a set of file descriptors to be monitored for writability.
error_fds	A pointer to a set of file descriptors to be monitored for errors.
timeout	A pointer to a struct timeval specifying the maximum time to wait for any of the descriptors to become ready. If NULL, select will block indefinitely until an event occurs.

Return

-Returns the total number of file descriptors that are ready and contained in the three returned descriptor sets.
-Returns -1 on error, with errno set to indicate the specific error condition.

Pseudo Code

```
Function select_descriptors(max_sd, read_fds, write_fds, error_fds,
timeout):
    Set up the read, write, and error descriptor sets.
    Clear all descriptor sets.
    Add descriptors to the appropriate sets using FD_SET macro.

    Call select with the parameters:
        - max_sd + 1 as the highest-numbered file descriptor plus one
        - read_fds, write_fds, and error_fds as the file descriptor
sets to monitor
        - timeout for maximum time to wait for an event

    If select returns an error:
        If the error is not due to interruption:
            Print an error message.
        Continue to the next iteration.

    Return the number of ready descriptors.
```

open_file

Parameters

Parameter	Description
path	file path

Return

-Returns file descriptor

Pseudo Code

```
Attempt to open the requested file.
Return the file descriptor or an error.
```

select

Parameters

Parameter	Description
max_sd	maximum descriptor number
read_fds	file descriptor set

Return

- Returns number of descriptors ready.

Pseudo Code

```
Monitor multiple file descriptors to see if any of them is ready for
IO operation using `select()`.
```

read_request

Parameters

Parameter	Description
sd	socket descriptor
buffer	storage buffer

Return

-Returns number of bytes read

Pseudo Code

```
Read data from the socket descriptor into a buffer using `read()`.
```

parse_request

Parameters

Parameter	Description
buffer	request buffer

Return

- Returns parsed request information (method, path).

Pseudo Code

```
Extract the HTTP method and path from the request buffer.
Determine the type of request (GET, POST, HEAD).
```

GET_request

Parameters

Parameter	Description
char *path	Path to the requested file or resource.
int client_socket	Socket descriptor for the client connection.

Return

-Doesn't return a value but sends a response directly to the client through the socket.

Pseudo Code

```
If the request is GET:
  Find the requested file.
  Send HTTP response headers and file content.
```

POST_request

Parameters

Parameter	Description
char *path	Path where the POST request is targeting.
char *body	The body of the POST request containing the data to be processed.
int client_socket	Socket descriptor for the client connection.

Return

-Doesn't return a value but sends a response directly to the client.

Pseudo Code

```
If the request is POST:
  Extract the POST data from the request.
  Process or store the POST data as needed.
  Send an appropriate HTTP response.
```

HEAD_request

Parameters

Parameter	Description
char *path	Path to the requested file or resource.
int client_socket	Socket descriptor for the client connection.

Return

-No return value, as it sends HTTP headers back to the client without a body.

Pseudo Code

```
If the request is HEAD:
  Find the requested file.
  Send only the HTTP response headers without body.
```

GET_response

Parameters

Parameter	Description
int client_socket	Socket descriptor for the client connection.
char *file_path	Path to the file being requested.

Return

-No return value. The function's purpose is to read a file and send its contents along with appropriate HTTP headers back to the client.

Pseudo Code

```
Check if the requested file exists and is accessible.
If yes:
  Open the file.
  Read the file's content into a buffer.
  Send HTTP status line "HTTP/1.1 200 OK".
  Send Content-Type header based on the file type (e.g., "Content-
Type: text/html" for HTML files).
  Send Content-Length header with the file size.
  Send a blank line to indicate the end of the headers.
  Send the content of the file read into the buffer.
Close the file after sending the content.
```

HEAD_response

Parameters

Parameter	Description
int client_socket	Socket descriptor for the client connection.
char *file_path	Path to the file being checked.

Return

-No return value. Sends HTTP headers similar to those in a GET request but without the body, indicating the file's existence and metadata.

Pseudo Code

```
Check if the requested file exists.
If yes:
  Send HTTP status line "HTTP/1.1 200 OK".
  Send Content-Type header based on the file type.
  Send Content-Length header with the file size.
  Send a blank line to indicate the end of the headers.
If not:
  Follow the 404_response pseudo code, but without the body.
```

POST_response

Parameters

Parameter	Description

int client_socket	Socket descriptor for the client connection.
char *response_body	The body of the response to send back to the client, typically a confirmation message or result of the POST operation.

Return

-No return value. Sends a response back to the client including the operation result.

Pseudo Code

```
After processing the POST request data (e.g., storing data in a database):
  Send HTTP status line "HTTP/1.1 200 OK" if the data was processed successfully.
  Send Content-Type header "Content-Type: text/plain" or another appropriate type based on the response content.
  Prepare the response body with the result of the POST operation (e.g., a confirmation message).
  Send Content-Length header with the length of the response body.
  Send a blank line to indicate the end of the headers.
  Send the response body with the operation result.
```

404_response

Parameters

Parameter	Description
int client_socket	Socket descriptor for the client connection.

Return

-No return value. Sends a 404 Not Found HTTP response to the client.

Pseudo Code

```
If the requested file does not exist or is not accessible:
  Send HTTP status line "HTTP/1.1 404 Not Found".
  Send Content-Type header "Content-Type: text/html".
  Send a blank line to indicate the end of the headers.
  Optionally, send a simple HTML document stating that the requested resource was not found.
```