Design

Overview:

So the design of this program is fairly simple. So the design is broken up into pieces as described below this is also the order in the way the code is designed:

- Head function
- Get function
- Put function
- Strtouint64 function(included in starter code)
- Create_listen_socket function(included in starter code)
- Handle_connection function
- Main function(included in starter code)

These functions make up the HTTP server program. I created separate functions for each request because this will make the code more legible to read and create a more organized code. Below I will describe the function and the way I designed them.

Head function

- This function handles the head requests
- It first checks if the file requested has the right permissions and if it is even located in this directory, if not it will return either a 403 forbidden message or 404 file not found the message to the client.
- After it initializes a length integer this will count the number of bytes read for the response to the client.
- Then there is a loop so it reads bytes from the file till it reaches the end.
- I used a buffer of 1000 this allows for a fast read but not any memory issues
- In the end, it will return a response message that the file is there and the length of the file requested.

Get function

- This function handles get requests.
- This design is fairly similar to the head request function besides the fact the function also has to send the file to the client.
- The first half of the function is the same as the whole head function.
- After this function has looped through the file to calculate the number of bytes, it will send the header for the get response so the client knows the file is there and how many bytes they will be expecting.
- Then the get function will reopen the file and start reading the bytes but instead of counting the bytes it will use send() to send the bytes of data read from the file directly to the client, which will be the body of the Get request. The buff will be dynamically allocated, with the type set an int so binary files can also be read.
- When it finished reading the file it will close the file.

Put function

- This function is drastically different from the head and get function.
- The function starts off with deleting the file if it is going to overwrite the file. This only happens if the file is there.
- Then the length variable which is the length of the file that is given through the function, sent by the client, will be converted to an integer because it is a string when given in the function.
- Then using open() with flags O_RDWR | O_CREAT, 0666, a new file will be created, setting permissions so it can be read and written to.
- Then a buffer is allocated using calloc to set the buffer to all zeros. The type is an int so binary files can also be read.
- And int r is initialized which is where the amount of byte read is going to be stored
- Then a while loop goes till bytes is less than zero, which is when all the bytes are read using recv() and written to the new file.
    - Every loop r which is the number of bytes read by the client's file gets subtracted from bytes which are the total bytes that the client sent.
    - Checks if r ever returns anything less than 1 meaning that connection has lost and closes and deletes the file and returns to handle_connection and closes the connection, going back to listening.
- Then it frees the buffer, closes the file, and eventually sends a 201 created response header to let the client know that the new file has been created or overwritten with the file that the client sent.
- The response is sent using a buffer called response which has been dynamically allocated this is to prevent memory from being leaked and memory overflow.

Strtouint64 function

- This code was given in the start code.
- It converts a string to a 16 bits integer.
- This is called the main function of the code.
- It is used to convert the given port number into a 16-bit number since to create a listening socket the port has to be in 16-bit format.

Create_listen_socket function

- This code was given in the starter code.
- But get called in the main function to create a listen socket, opening up for clients.
- It uses the 16-bit port number and creates the socket and sets the address and port for the socket
- Returning a listen file descriptor

Handle_connection function

- This function handles the client's header parsing and calling the right request function. Also checking if the message sent by the client is okay.
- It first allocates memory for the following variable: host, HTTP version, request type, file, temp file, and buff for reading the client's header. This is dynamically allocated to prevent the overflow of memory.
- There is an infinite loop around the following code because if there is an issue, the code uses the break statement to break out, this allows for fewer lines of code to free the allocated memory, instead, it is outside of the while loop so when there is any issue or reaches the end there is a break statement.
- Within the infinite loop
    - It first reads the header sent by the client
    - Using sscanf it parses through the buff which holds the bytes for the client's header
    - It checks if the file given by the client has the allowed characters
        - It does this by looping through each character and checking using isalnum() to check if it is a number or a letter. It also allows for the characters '.' and '_'.
        - If a character is not any of the allowed characters it will set a variable check_file to 1 and break the loop
    - Then it loops through the given host
        - It loops through the string checking if there are no spaces
        - If there is a space, it will set a variable check_host to 1 and break the loop
    - After that, it creates a temp HTTP version variable that is set to the required HTTP version which is HTTP/1.1.
    - Then it uses an if statement to check :
        - If the given HTTP version is not the same as the variable we allocated space for and set the required HTTP version
        - If either the check_host or check_file are set to 1
        - Or the length of the file is smaller than 2 or bigger than 20 and does not start with '/'
        - If any of these are satisfied it will send the client a 400 bad request response
    - Then, I have for loop that loops 19 times so that it copies the file given to the temp file holder where it copies the whole string besides the first one:

```
    for(int i = 0; i < 19; i++){
      f[i] = file[i+1];
    }
```

- Then it creates three variables that are set to “GET”, “HEAD”, and “PUT”
- This is so the program can use strcmp() to compare the given request type to those variables so the code know what function to call
- After creating the variables it will check if any of the variables are equal to the request given by the client
    - If so it will call that request function
    - If it is put it will scan the buffer one more time to get the content length of the file sent by the client
    - Else it will send a 500 not implemented message header to the client saying that the given request type is not implemented
- Then there is a break statement to break out of the loop

- When the while loop is excited all the dynamically allocated memory is freed and the connection between the client and the host is closed.

Main function

- This was part of the given starter code
- This is the main function where the port is set and listen_socket gets called to start listening for connections.
- After setting up the socket and the socket started listening, there is a infinitely loops listening till it connects with a client and calls handle connection, which handles the request made.
- After the connection is closed and returned from the handle_connection function, the code loops till it get another client connection.