PA04 Proxy Creation Write Up
Name: Adam Caudle
ID: 011774063

Intro:
For this assignment, I created a basic web proxy that handles HTTP 1.0 traffic and re-directs it based on user input. This web proxy would run as a server, take requests, parse them, build headers, forward the traffic, and then relay the response. It has been tested and works with Netcat, telnet, curl, and Firefox.

Implementation:
Main:
For the main function, I referenced how the Tiny web-server code works to get an idea of the structure. My main ensures that the user specifies a port number, opens a listening socket, and then runs an infinite loop. While that loop is running, anything sent into the server is captured and put through a function that performs the request on behalf of the traffic entering.

Perform_proxy_request:
To perform a request, my proxy server had a couple of steps. First, it initializes and reads into a robust I/0 buffer (RIO) to perform networking operations. It then checks to make sure the request is a compatible method and HTTP version before it makes its request. If it isn't it prints that it's unsupported and continues server operation. Then, the function parses the URI to get the correct information into the correct character buffers and calls a function to forward the request.

Parce_uri:
To parse the URI, my function checks the first couple of characters to see if "http://" is included in the path. If it is, it removes it and enters a loop to grab the host IP/name. After hitting the end of the host, it checks for a ":" to see if a port number is included. If it is, it grabs the port number and stores it. If not, the port number is set to the default port 80. Then, the rest of the path is copied for navigating directories on any server the request is sent to. With all of these variables stored, the request is ready to be forwarded.

Forward_request:
My forwarding function starts by opening up a connection between the proxy and the new server. After this, it builds the header using the build_http_header function. This allows it to form a request for it to send. After this, it writes to the proxy's new client all the information and waits for all the response information to come through. After this all comes through, the information is written back to the original requester. This ensures that the connection is opened by the proxy server and sent back to the proxy's original client.

Build_http_header:
To build my http_header, I first specify the request type (as GET for this assignment). I then add the query path, and end with the version and a termination character. Then, some other information is written on new lines below the request so that there is more information from the proxy. This is useful for identification and debugging.

Conclusion:
The implementation of this web proxy successfully demonstrates the principles of proxy server operations, including parsing HTTP requests, forwarding them to target servers, and relaying responses to clients. By leveraging robust I/O handling and precise URI parsing, the proxy ensures compatibility with various tools like Curl and Firefox. This project showcases a solid understanding of network programming concepts and highlights the importance of structured request handling and debugging-friendly design.

QUESTION:
Why are some URLs not accessible?
Some URLs are not accessible due to incompatibility with the proxy. The proxy only handles HTTP 1.0 and GET, so HTTPS versions or requests using other HTTP methods can cause errors. With more time, building out the proxy could mitigate these shortcomings.