Jordan Mai
1645861

1. **Project Description:**
    a. Our program that we are creating will serve as a reverse proxy for a server and client. The proxy will represent a middleman that intercepts all connections that come from the server and client. If the request from the client is a “PUT” or “HEAD”, the proxy will just take in the request and deliver it to the server and then receive that response and deliver it back to the client. If the request is a “GET”, the proxy will receive the request and then check with the server to see the date of modification from the server by requesting a “HEAD” response. The program is also responsible for caching responses and its data. By caching, it will save time from request and response between client & server because the proxy already has the data ready to be sent back, only if the file is the most up to date. If either a newer modified version has been returned or if the cache contains the response, we will add to cache. If the cache is full we will replace one based on the rule determined by the user.

2. **Program Logic - PSEUDOCODE**
    a. **Cache Data Structure:**
        i. **Cache *c;**
        ii. **c->array[allocated memory] = buffer to contain response**
        iii. **c->tag[] = uri for the file body**
        iv. **c->time = time of date modification**
        v. **c->length = file size**
        vi. **Struct cache *next;**
    b. **Int main()**
        i. getopt(if the flags in the argument is -c or -u or -m):
            1. If (c):
                a. Max cache size
            2. if(u):
                a. Triggers LRU for caching
            3. If (m):
                a. Max file size
            4. Handle Port Number >1024

        ii. Initialize cache for buffer

        iii. while(1):
            1. Accept connection
            2. Connect to server
            3. handle_connection(connfd,servfd);
        iv. Close(connfd)

c. **Handle_connection**
    i. While (able to receive bytes from client curl using recv()):
        1. Receive the request line and store in a buffer
        2. Parse each word separated by space. To obtain the request, body of file, version, content length and content. Then store each in a separate buffer.
        3. IF(request is not correct):
            a. Send 400 error
            b. Close connection
        4. IF(strcmp(request is “GET”)):
            a. Send to helper function for “GET”
        5. IF(request is “PUT”):
            a. Send to helper function for “PUT”
        6. IF(request is “HEAD”):
            a. Repeat “GET” but only send header and not body for response
    ii. Close(connection)

d. **Handle_get(connfd, object):**
    i. IF(exists a file that is cached):
        1. Create HEAD request
        2. send(serverfd)
        3. recv(serverfd)
        4. Check the time of modification on the response from server
        5. if(the received response has newer time, then:
            a. write_cache() function
        6. Else:
            a. read_cache() to obtain cached response
        7. send(connfd) the response
    ii. Else:
        1. send(serverfd)
        2. recv(serverfd)
        3. write_cache()
    iii. Close file

e. **Handle_put(connfd,object):**
    i. Receive all bytes from client
    ii. Send request to server
    iii. Recv from server
    iv. Send back to client
    v. Close file
f. **Handle_head(connfd,object):**
    i. Receive all bytes from client

ii. Send request to server
iii. Recv from server
iv. Send back to client
v. Close file

g. **write_cache(cache *c):**
    i. If cache is not full based on max size:
        1. Create a node
        2. Store information of request to empty slot
    ii. If cache is full:
        1. If LRU:
            a. For range in beginning of cache to end:
                i. Determine the least recently used(or called)
                ii. Replace data with newer one
        2. if(FIFO):
            a. Ptr = head of cache
            b. while(head !end of cache):
                i. head= head->next;
            c. Replace data with newer one
h. **read_cache():**
    i. Find the correct uri form the cache
    ii. Retrieve pointer and replace data

i. **check_caache():**
    i. Loops through whole cache to check for existence of a uri
    ii. Returns the pointer to that node

3. **Data Structures**
    a. **Array-** An array is used to store some byte amount to be used. We will use an array buffer to store bytes taken in from the client and also use arrays to write information into. The array needs to contain all the information and is also used to build a string of response headers.
    b. **Files -** A file is used to put bytes of content into to be kept outside of a program or in this case to be sent over to users. For this program, a file is used to read and write bytes from the client server.
    c. **Struct linked list-** a struct of linked list is used to store multiple data on what pointer. For this program, we use a linked list to easier form our cache and store multiple properties to one uri. Using a pointer to the next object is easier to use.

4. **Functions**
    a. **Int main(int argc, int argv[]):**
        i. This function opens a socket and attempts to connect to the same port as a client server. This function checks for any errors when reading from

argv[] and makes sure that all arguments are correct. It will persist for a while loop to keep waiting and receiving connection until closing. The error to check for is the correct amount of argument inputs and if connection is possible. This function is also responsible for connecting to a server.

b. **Strtouint16(char numer[]):**
    i. Converts a string to a 16 bit unsigned integer
    ii. Return 0 if string is out of range
c. **create _listen_socket(unit16_t port):**
    i. Creates a socket for listening to the port
    ii. Closes and creates an error message on failure
d. **create_client_socket():**
    i. Creates a socket for connecting to port to a server

e. **Handle_connection(int connfd):**
    i. This function receives every byte that is sent from the client server. It will continue to keep receiving. This function will handle all necessary requests (“GET”, “PUT”, “HEAD”) only . This function will parse through the entire buffer and take out necessary elements to be put into different buffers. The function continues with a while loop until everything is received from the client side. This program also checks for errors such as “400 Bad Request” if the request that was sent doesn’t match the appropriate components, or “501 Not Implemented if the request is properly formatted but request is valid but not necessary for this assignment. If everything works, this function will send separate helper functions for each different request. Generate a “500 Internal Server Error” if an error occurs on the program. This will send to other functions.

f. **send_get(int connfd,char*body,char*version):**
    i. This function will be responsible for handling “GET” requests. This will be responsible for retrieving and sending information back and forth between client and server. This function will also be in charge of handling cache inputs/outputs and any other request building to retrieve information from a client or server.

g. **send_put(int connfd, char*body, char*version, char* content_num):**
    i. This function will be responsible for handling “PUT” requests. This function is responsible for sending and receiving requests and responses if they are PUT related.
h. **send_head(int connfd, char*version, char*body):**
    i. Same as send_put but for head requests.
i. **Void write_cache(cache *c):**

i. This function is responsible for writing the uri and request in a cache linked list. This is also responsible for checking for errors and determining which slot should be added or replaced.

j. **Void read_cache(cache *c):**
   i. This function just reads information from a pointer to a cache node and is also in charge of replacing that specific node

k. **Void cache_check(cache *c):**
   i. Checks if a particular uri exists in the stored cache

l. **Void error_check:**
   i. Checks for any 400 or 404 errors and sends back to client without going to server

5. **Using a large file (e.g. 100 MiB — adjust according to your computer's capacity) and the provided HTTP server:**

   Start the server with only one thread in the same directory as the large file (so that it can provide it to requests);
   Start your proxy with no cache and request the file ten times. How long does it take?

   I made a large file with size 10000000000 bytes and ran the proxy server with 0 size cache. It took around 5-10 seconds to complete the sending and receiving from client and server, for each run of the ten times. Since I am not caching the files, for every time I run the proxy, it will always send to the server and receive from the server every single byte, even though nothing was modified from the file. This is very inefficient and wastes time.

   Now stop your proxy and start again, this time with cache enabled for that file. Request the same file ten times. How long does it take?

   With a cache, my proxy increased significantly for sending and receiving. The first time obviously took longer, with the usual 5-10 secs for completing proxy. But with a cache, it processed in 1-2 seconds. This was way faster to send and receive, because I already stored all content and requests from the 100 MiB to memory, so all I had to do was skip the step to send to send and receive from server, and I could already send my content back to client.. My proxy saved time from not having to send to the server and receive from the server every single byte for all run times even though it contained the same content.

   Aside from caching, what other uses can you consider for a reverse proxy?

   A reverse proxy can be used to check for errors, so we won't have to send it to a server if the request is already bad.
   We can also use a reverse proxy to check for viruses or bad information that is trying to be sent into the server or client. This proxy can act as a protection and verification to see if everything is normal. Besides protection, a proxy is used for privacy, as it can alter

information from what is being sent, so the other end won’t be able to access any information if wanted to implement.