

## **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**

### **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
- 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:
    - 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
        - ii. Replace data with newer one
    - 2. if( FIFO):
      - a. while(!end of cache):
        - i. while->next;
      - b. 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

### 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. Strtoint16(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(uint16\_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
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?  
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?**

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