Mark Moreno maalmore@ucsc.edu	Asgn_1 DESIGN.pdf

```
GLOBAL VARIABLES:
Int log_file
Int server_sockd
Int log_request
Int entries
Int errors
Int counter
char* port
```

I initialize all of these as global variables because they will need to be passed around in threads

```
Int substring_index(char *string, char *sub_string):
        |
        If (string.size()) >= (sub_string.size()):
                |
                for ( i = string.size() - sub_string.size()) to 0):
                        |
                        for (j = 0 to sub_string.size()):
                                |
                                If (string[ i + j ] != sub_string[ d ])
                                        |
                                        NOT FOUND
                        |
                        If (FOUND)
                                |
                                return i
                |
                If (NOT FOUND) return -1
        |
        else:
                |
                return -1
```

The purpose of this substring_index() function is to find substrings within larger strings. This function will require a (char* string) and a (char* substring) and return the index position of the substring within the string. It will return -1 otherwise

```
char* get_substring( char string , int starting_index, char end):
        |
        Int index = starting_index
        while( string [index] != end ):
                |
                Index++
                length++
        char substring[ length ]
        count = 0
        index = starting_index
        while( string [index] != end):
```

get_substring() will return a substring within a larger string. You will need to know the index of the first character of the substring within the string. You will also need the character that comes after the final character inside the substring.
The string that belongs within the starting index and the last character will be stored in a char* and returned.

```
                |
                substring[count] = string[index]
                Index++
                count++
        return substring
```

```
TEST ERROR 400(header, filename):
        |
        Int err = 0
        Char http = get_substring(header, 7+len(file_name), '\r')
        if(len(file_name)>27 or len(file), 'r')err = 1:
        Else:
                |
                For: i = 0 to len(file_name):
                        |
                        if(file_name)contains:
                                ('a' to 'z' or
                                'A' to 'Z' or
                                '0' to '9' or
                                '-' or '_')
                        Else:
                                |
                                 err = 1
                                break
        |
        If (err = 1):
                |
                send ("HTTP/1.1 400 Bad Request\r\n)
                write(err)
                if(log_request=1):
                        |
                        pwrite(log_file, "FAIL: <req> /<file_name>
                        HTTP/1.1 --- response 400\n========\n)
                errors++
                if(count<total threads) :cond_signal( cond_for)
                else : cond_signal ( cond )

                Mutex_unlcok ( mutex1 )
                Return 0
```

```
TEST ERROR 404(file):
        if(file<0):
                |
                send(HTTP/1.1 404 Not Found\r\n)
```

```
                    write(err)
                    if(log_request=1):
                              |
                              pwite(og_file, “FAIL: <req> /<file_name>
                              HTTP/1.1 --- response 404\n========\n)
                    |
                    errors++
                    if(count<total threads) :cond_signal( cond_for)
                    else : cond_signal ( cond )

                    Mutex_unlcok ( mutex1 )
                    Return 0
```
```
CHECK_HEALTHCHECK_NOT_GET(file_name):
          if(file_name = “healthcheck”):
                    |
                    Char err_mes = “HTTP/1.1 403 Forbidden\r\n”
                    send( err_mes )
                    write( err_mes)
                    if(count<total threads) :cond_signal( cond_for)
                    else : cond_signal ( cond )
                    Mutex_unlcok ( mutex1 )
                    Return 0
```
```
CHECK_HEALTHCHECK_GET(file_name):
          |
          if(log_request = 1):
                    |
                    Entries++
                    HEALTH = “HTTP/1.1 200
                    OK\r\nContent-Lenth:\r\n\r\n <errors>\n<entries>”
                    send(HEALTH)
                    write(HEALTH)
                    if(count<total threads) :cond_signal( cond_for)
                    else : cond_signal ( cond )
                    Mutex_unlcok ( mutex1 )
                    Return 0
          Else:
                    |
                    send(err_404)
                    write(err_404)
                    Errors++
                    if(count<total threads) :cond_signal( cond_for)
                    else : cond_signal ( cond )
                    Mutex_unlcok ( mutex1 )
```
LOG_TO_FILE: <<<not a function>>>	

|
Char hex
Char zeros = “00000000”
for(i = 0 to byte):
    |
    if(a%20=0):
        |
        pwrite(log_file, hex)
        sprintf(byte_size, “%ld”, a)
        Int inc = 0;
        for(j = 0 to 8):
            |
            if(j>=(8-len(byte_size))):
                |
                zeros[j]=byte_size[inc]
                inc++
        hex+=zeros
    |
    sprintf(temp, “%02x”, buff[i]);
    hex+= “ <temp>”
    a++	

Void server_thread:
    |
    mutex_lock(mutex1)
    Counter++
    cond_signal(thread_create)
    cond_wait (server_wait, mutex1)
 	
    struct client_address (2)
    socklen_t client_address_lenght (3)
    print server is waiting
    Int client_socket = accept ((1) ,&(2), &(3))
    uint8_t buff [Buffer_size + 1]
    ssize_t bytes = receive client_socket
    buff [bytes] = null terminate
    print received bytes from client plus response
 	
    for(int i = 0 to bytes):
        char Header[i] = buff [i]
    char request = get_substring(header, 0, ‘ ‘)
	**==
	=**
	
	Starter code
	
	**==
	=**
	We save the header to a separate char
	
	We use helping function get_substring to get request

If (request == “PUT”) :
    Index = substring_index(header, “Length: “)
    char len = get_substring(header, index+6, ‘\n’)
    char filename = get_substring(header, 6, ‘ ‘)
    CHECK HEALTHCHECK_NOT_GET
    CHECK 400
    CHECK 404
    file = open(filename)
    while((bytes = recv(buff))>0):
        |
        write(file, buff, bytes)
    close file
    send(client_sockd, response)
	Use helper functions to parse data
	
	A while loop is used to read received data and store it into buff and at the same time write data into our file.
	
	Once everything is completed we send a response to the client

Else if (request == “GET”):
    char filename = get_substring(header, 6, ‘ ‘)
    file = open(filename)
    char data[buffersize]
    CHECK HEALTHCHECK
    CHECK 400
    CHECK 404
    Buffer = lseek(file)
    char len=”Buffer”
    response = (“HTTP/1.1 200 OK\r\nContent-Length:” + len + ”\r\n\r\n”)
    send(reponse)
    while(byte = read(file, data, buffersize)>0):
        |
        send(client_sockd , data, byte)
        LOG_TO_FILE
    close file
	If file does not open, we must send a 404 response
	
	We obtain length of file and convert the value into string to append to response
	
	Response is created and sent to client and will be followed by data from file
	
	While loop is used to read data from file buffersize at a time and send the data to the client

Else if (request == “HEAD”):
    char filename = get_substring(header, 7, ‘ ‘)
    file = open(filename)
    char data[buffersize]
    CHECK HEALTHCHECK_NOT_GET
    CHECK 400
    CHECK 404
    Buffer = lseek(file)
    char len=”Buffer”
    response = (“HTTP/1.1 200 OK\r\nContent-Length: ” + len + ”\r\n\r\n”)
    send(reponse)
    close file
	Same as “GET” except data is not sent

Else:
        |
        Char err = “HTTP/1.1 400 Bad Request\r\n “
        send(err)
        write(err)
        if(log_request==1):
            |
            Char err_msg = “FAIL: /<file_name>
            HTTP/1.1 --- response 400\n=====\n”)
            pwrite(err_msg)
        |
        errors++
	Send bad request if header does not contain correct content

main():
    |
    |
    log_request=0
    for(i = 0 to argc):
        |
        if(argv[i]=”-N”):
            |
            i++
            Threads = argv[i]
        |
        else if(argv[i]=”-l”
            |
            I++
            logfile = “argv[i]
            log_request =1;
 	
        |
        else port = argv[i]
    |
    char *argument_1
    struct socket_address_in
    struct server_address
    memset (server_address)
    int server_socket = create server socket (1)
    if (server_socket < 0) print **error**
    int enable = 1 **>>>**
    int ret = avoid: `'Bind: Address Already in Use'`
	
	Starter code

```
ret = bind server address to open socket
ret = listen for incoming connections
if ( ret < 0 ) end

mutex_lock(mutex3)
for(int i = 0 to thread):
        |
        create_thread(thread[i], thread_server)
        cond_wait(thread_create, mutex3)
|
mutex_unlock(mutex3)

mutex_lock(mutex)
for(int j to thread_count):
        |
        cond_signal(server_wait)
        cond_wait(cond_for, mutex)
|
mutex_unlock(mutex)

mutex_unloxk (mutex2)
while(1):
        |
        create_thread( thread[i], thread_server)
        Cond_wait (cond, mutex)
        i++
|
mutex_unlock(mutex2)
```
```
close(client_sockd)
return 0;
```