

Assignment 3 Design Document

Alan Caro
CruzID: alcaro

CSE130, Fall 2019

1 Goal

The goal of this program is to make our HTTP server support caching.

2 Assumptions

I am assuming the user will only use the curl command. No directories will be sent by the client or requested. When no Content-Length is passed the client will have to exit the process by himself/herself. Also, I am assuming the user will send request with this format:

PUT:

```
curl -T localfile http://localhost:8080 --request-target filename_27_character
```

GET:

```
curl http://localhost:8080 --request-target filename_27_character
```

3 Design

The general approach I am taking is to check what request and file name the user passed. Then, if the request and file name is valid, I open a socket then process the GET or PUT command accordingly. If there is any errors like an invalid file or request I send the appropriate error code.

To handle caching, I followed the FIFO algorithm. I created a Cache class that has an `std::list` files as an attribute. `files` is a list that contains a struct that holds a files information such as file name, contents, size, and if the file is dirty. The Cache class has methods that insert files to the cache, display the files in the log file, check if a file exists in the cache, modify a file, remove a file from the cache, and a method to send the file to the client for GET requests.

4 Pseudocode

This is the core pseudocode for the program. Note that it's pseudocode, not C (or Java or Python) code.

```

class Cache
    method insert
    method display
    method hasFile
    method setDirty
    method remove
    method sendContents
    Declare std::list files

```

```

procedure main
    Declare string hostname
    Declare string port
    Declare option

```

```

loop
    switch(option)
        case 'c':
            ISCACHING = 1
            break;
        case 'l':
            LOGFILE = optarg
            break
        case ':':
            fprintf("option needs a value\n")
            break
        case '?':
            fprintf("unknown option: %c\n",optopt)
            break

```

```

hostname = argv[argc - 2]
strcpy(port, argv[argc - 1])

```

```

if(LOGFILE)
    FD_LOG = open(LOGFILE, O_CREAT | O_RDWR | O_TRUNC, 0644)

```

```

Declare struct addrinfo addrs, hints
hints.ai_family = AF_INET
hints.ai_socktype = SOCK_STREAM
getaddrinfo(hostname, port, &hints, &addrs)
main_socket = socket(addrs.ai_family, addrs.ai_socktype, addrs.ai_protocol)
enable ← 1
setsockopt(main_socket, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(enable))
bind(main_socket, addrs.ai_addr, addrs.ai_addrlen)

```

```
listen(main_socket, 16)
```

```
loop
```

```
    Declare client_socket = accept(main_socket, NULL, NULL);  
    processOneRequest(client_socket)
```

```
return 0
```

```
procedure processOneRequest
```

```
    Declare request  
    Declare fileName  
    Declare buffer
```

```
    memset(buffer, 0, 1024)  
    recv(socket, buffer, 1024, 0)  
    sscanf(buffer, "%s %s", request, fileName)
```

```
    if fileName[0] == '/'  
        memmove(fileName, fileName + 1, strlen(fileName))  
    if isValidName(fileName) == -1
```

```
        if(LOGFILE)
```

```
            Declare message
```

```
            sprintf(message, "FAIL: %s %s HTTP --- response 400\n=====\n",  
request, fileName)
```

```
            printLog(message)
```

```
            send(socket, "HTTP/1.1 400 Bad Request", strlen("HTTP/1.1 400 Bad  
Request"),0)
```

```
            return
```

```
    if strcmp(request, "GET") == 0  
        processGet(fileName, socket)
```

```
    else if strcmp(request, "PUT") == 0
```

```
        line ← strtok(buffer, "\r\n")
```

```
        array[7]
```

```
        word[20]
```

```
        i ← 0
```

```
        j ← 0
```

```
        loop
```

```
            array[i++] = line
```

```
            line = strtok(NULL, "\r\n")
```

```

        loop
            if strstr(array[j], "Content-Length: ") != NULL
                break

            if array[j] != NULL
                sscanf(array[j], "%*s, %s", word)
                i = atoi(word)
            else
                i = -1
            processPut(fileName, socket, i)
        else
            if (LOGFILE)
                Declare message
                sprintf(message, "FAIL: %s %s HTTP --- response 400\n=====\n",
request, fileName)
                printLog(message)

                send(socket, "HTTP/1.1 400 Bad Request", strlen("HTTP/1.1 400 Bad

```

```

procedure isValidName
    Declare variable j
    Declare struct path_stat
    loop
        j += 1
        if j != 27
            return -1
    loop
        c ← fileName[i]
        if isalpha(c)
            continue
        if isdigit(c)
            continue
        if c == '-'
            continue
        if c == '_'
            continue
        return -1

    return 0

```

```

procedure processGet
    fd ← open(fileName, O_RDONLY)

```

```

buffer[32]

if fd == -1
    if access(fileName, F_OK) == -1
        if (LOGFILE)
            Declare message
            sprintf(message, "FAIL: %s %s HTTP --- response
404\n=====\\n", request, fileName)
            printLog(message)

            send(socket, "404 Not Found" strlen("404 Not Found"),0)
        else
            if (LOGFILE)
                Declare message
                sprintf(message, "FAIL: %s %s HTTP --- response
403\n=====\\n", request, fileName)
                printLog(message)

                send(socket, "403 Forbidden" strlen("403 Forbidden"),0)
            return

Declare struct of type stat st

if stat(fileName, &st) == 1)
    send(socket, "500 Internal Server Error" strlen("500 Internal Server Error"),0)

size ← st.st_size

if(LOGFILE)
    Declare message
    if(ISCACHING)
        Declare file f
        f.name ← (char*)malloc(strlen(fileName))
        strcpy(f.name, fileName)
        Declare bool fileInCache = CACHE.hasFile(fileName)

        if(fileInCache)
            sprintf(message,"GET %s length 0 [was in cache]\\n=====\\n",
fileName)
            CACHE.sendContents(fileName, socket)
        else
            f.size ← size
            Declare string str

```

```

        sprintf(str, "HTTP/1.1 200 OK \r\nContent-Length: %d\r\n\r\n",
size)
        send(socket, str, strlen(str), 0)

        Declare string result
        loop
            Declare string buf
            result.append(buf)
            send(socket, buffer, 1, 0)
        f.contents = result
        CACHE.insert(f)
        sprintf(message, "GET %s length 0 [was not in
cache]\n=====\n", fileName);

        Else
            loop read(fd,buffer,1)
                send(socket, buffer, 1, 0)

        sprintf(message, "GET %s length 0\n=====\n", fileName);

        printLog(message)
        close(fd)

procedure processPut
    Declare fd
    Declare string buffer

    if(!LOGFILE && !ISCACHING)
        fd = open(fileName, O_CREAT | O_RDWR | O_TRUNC, 0644)
        Declare i = 0
        loop
            read (socket, buffer, 1)
            write(fd, buffer, 1)
            i++
        send(socket, "HTTP/1.1 201 Created \r\nContent-Length: 0\r\n\r\n",
strlen("HTTP/1.1 201 Created \r\nContent-Length: 0\r\n\r\n"), 0);
        return

    Declare bool fileInCache = CACHE.hasFile(fileName)

    if(ISCACHING)
        if(!fileInCache)

```

```

    Declare i = 0
    loop
        read (socket, buffer, 1)
        write(fd, buffer, 1)
        i++
    close(fd)

```

```

Declare string result
Declare file f
f.name = (char*)malloc(strlen(fileName))
f.size = size
strcpy(f.name, fileName)

```

```

fd = open(fileName, O_RDWR)
loop
    read(fd, buffer, 1)
    Declare string buff
    f.contents.append(buffer)

```

```

CACHE.insert(f)
else
    CACHE.setDirty(fileName, socket, size)

```

```

if (!LOGFILE)
    send(socket, "HTTP/1.1 201 Created \r\nContent-Length: 0\r\n\r\n",
    strlen("HTTP/1.1 201 Created \r\nContent-Length: 0\r\n\r\n"), 0);

```

```

fd ← open(fileName, O_RDWR)
Declare buffer_log
Declare address
Declare target
Declare length

```

```

if (LOGFILE && ISCACHING)
    if(fileInCache)
        length += sprintf(target + length, "PUT %s length %d [was in cache]\n",
        fileName, size)
        printLog(target)
        CACHE.display(fileName)
    else
        length += sprintf(target + length, "PUT %s length %d [was not in
        cache]\n", fileName, size)

```

```

loop
    read(fd,buffer_log,20)
    length += sprintf(target + length, "%08d ", address);
    loop
        length+= sprintf(target + length, "%02x ", buffer_log[j])

        length+= sprintf(target + length, "\n")
        printLog(target)
        length = 0
        address+=20
    close(fd)
    Declare buff
    strcpy(buff, "=====\n")
    printLog(buff)
send(socket, "HTTP/1.1 201 Created \r\nContent-Length: 0\r\n\r\n",
    strlen("HTTP/1.1 201 Created \r\nContent-Length: 0\r\n\r\n"), 0);

else if(LOGFILE)
    Declare i = 0
    loop
        read (socket, buffer, 1)
        write(fd, buffer, 1)
        i++
    close(fd)

    fd = open(fileName, O_RDWR)
    length += sprintf(target + length, "PUT %s length %d\n", fileName, size)

    loop
        length+= sprintf(target+length, "%08d ", address)

    loop
        length+= sprintf(target + length, "%02x ", buffer_log[j])

        length+= sprintf(target + length, "\n")
        printLog(target)
        length = 0
        address+=20

    close(fd)
    Declare buff
    strcpy(buff, "=====\n")
    localOffset+= printLog(buff, localOffset)

```

```
send(socket, "HTTP/1.1 201 Created \r\nContent-Length: 0\r\n\r\n",  
strlen("HTTP/1.1 201 Created \r\nContent-Length: 0\r\n\r\n"), 0);
```

procedure printLog

```
    lineLength = strlen(message)
```

```
    write(FD_LOG, message, lineLength)
```