Deign Document: ASGN-4 Adding aliases to the HTTP server

Hang Yuan

CruzID: hyuan3

**1 Goals**

The goal of this project is to implement aliases functionality to support different name pointing to one HTTP name. The aliases will adopt KVS method to store data and in-memory cache to support better performance.

Both client and server now support PATCH command which accompany the alias KVS file to implement. Compared to data KVS file, alias KVS file don’t have data range that have be pointer by entries. It will be directly stored in KVS file.

Other than that, server now will delete the slash char ‘/’ for all names.

**2 Design**

There are three parts in the design: (1) client: support PATCH command; (2) server: alias key-value store implementation; (3) multithreading and concurrency control.

**2.1 Client: support PATCH command**

Client now support PATCH command in the form of “a:existing\_name:new\_name”. Where the new\_name is an alias to existing\_name.

The new\_name and existing\_name can be both alias names and existing\_name could be a real httpname. If either one is lacking in command line, client will print out error message.

1. **Input :** **buf**[4096]stored command arguments: **a:existing\_name:new\_name**
2. Obtain action, existing\_name, new\_name
3. **if** buf[0] == ‘/’ **then** // delete slash ‘/’
4. strncpy(httpname, httpname + 1, strlen(httpname))
5. **end**
6. **if** strlen(httpname) == 0 || strlen(filename) == 0 **then**
7. fprintf(stderr, "HTTP/1.1 400 Bad Request\n")
8. **continue**
9. **end**
10. **if** argv[i][0] == ‘a’ **then**
11. send(sock,“PATCH existing new\_name \r\n\r\n ALIAS existing new\_name \r\n)
12. **end**

Algorithm 1: Client side additional code to send PATCH command

**2.2 Server: alias key-value store implementation**

Server now will support alias functionality. To achieve this goal, there are four functions need to be added into the program: (1) alias KVS file and alias cache initialization; (2) server side: handling PATCH request; (3) name\_handler to support mapping alias to existing name; (4) changes in PUT and GET execution.

**2.2.1 alias KVS file and alias cache initialization**

Server now support “-m filename” to open or create alias KVS file. If it open a KVS file, then the program will call alias\_init() to reload entries into cache. If it creates a new KVS file, then do nothing.

**Input :** char \* **optarg** int32\_t **argc** char \* **argv[]**

**Shared Variable** **:** int32\_t **fd\_alias**

1. **while** (opt = getopt(argc,argv, “N:c:f:m:”) != 1 **do**
2. **switch**(opt)
3. **case** ‘m’:
4. **if** (fd\_alias = open(optarg, O\_RDWR)) == -1 **then**
5. **if** (fd\_alias = open(optarg, O\_CREAT | O\_RDWR | O\_TRUNC, S\_IRWXU | S\_IRWXG | S\_IRWXO)) == -1 **then**
6. fprintf("SET UP FAILED: cannot open or creat the KVS file\n")
7. **exit**(EXIT\_FAILURE);
8. **end**
9. **end**
10. alias\_init(); // reload alias kvs entries to cache
11. **end**
12. **done**

Algorithm 2: open or create alias KVS file in main function

In order to keep data consistency, the program will reload all entries back into the cache after open the alias KVS file. The program adopts in-memory map alias\_map<string, uint32\_t> for cache usage.

The cache will store two elements <entry, alias\_end>. The entry is char[128] contains exisiting\_name and new\_name with null terminator after each of two. The combined two names will two null terminators will not exceed 128 bytes. The alias\_end is a shared variable pointing to the end of alias KVS file.

**Input :** None

**Shared Variables :** int32\_t **fd\_alias** uint32\_t **alias\_end**

**function void** alias\_init()

1. char entry[128]
2. **while** pread(fd\_alias, entry, 128, alias\_end) != 0 **do**
3. alias\_map.insert(make\_pair(entry, alias\_end))
4. alias\_end += 128;
5. **done**

Algorithm 3: alias\_init() reloads all entries in KVS file back to cache

**2.2.2 server side: handling PATCH request**

When the request header contains a PATCH, the program will go into PATCH processing procedure. This procedure will first parse the request header and delete the / char if it exists in front of the name.

**Relevant Variables :** char \* **existing\_name** char \* **new\_name**

1. recv(cl, buf, sizeof(buf), 0)
2. ptr = strstr(buf, “ ”)
3. strncpy(action, buf, ptr - buf) // obtain action code
4. **if** strcmp(action, "PATCH") == 0 **then** // PATCH command
5. ptr\_tem = strstr(ptr + 1, " ")
6. **if** buf[ptr + 1 - buf] == '/' **then** // delete ‘/’ char
7. ptr += 1;
8. **end**
9. memcpy(existing\_name, ptr + 1, ptr\_tem - (ptr + 1)) // obtain existing\_name
10. ptr = strstr(ptr\_tem + 1, "\r\n\r\n")
11. **if** buf[ptr\_tem + 1 - buf] == '/' **then** // delete ‘/’ char
12. ptr\_tem += 1
13. **end**
14. memcpy(new\_name, ptr\_tem + 1, ptr - (ptr\_tem + 1)) // obtain new\_name
15. **end**

Algorithm 4: parse the PATCH request header and delete ‘/’ char

The procedure will first check if the new\_name which is alias exists in cache. If alias not found, the entry including “new\_name\0existing\_name\0” will be written into alias KVS file and will be updated in the cache. If alias found, the new entry will be written into the alias KVS file and leave cache unchanged.

To avoid data race, when the program writes entry into the KVS file, the mutex lock will control one thread can access alias\_end and write into the KVS file.

**Relevant Variables :** map<string, uint32\_t> **alias\_map**

**Shared** **Variables :** int32\_t **fd\_alias** uint32\_t **alias\_end** mutex **alias\_end\_lock** mutex **alias\_busy**

1. map<string, uint32\_t>::iterator iter;
2. memcpy(file\_buf, new\_name, strlen(new\_name));
3. memcpy(file\_buf+strlen(new\_name)+1, existing\_name, strlen(existing\_name));
4. **if** (iter = alias\_map.find(new\_name)) == alias\_map.end() **then** // not found
5. // store alias in KVS file and update cache
6. pthread\_mutex\_lock(&alias\_end\_lock);
7. pwrite(fd\_alias, file\_buf, 128, alias\_end);
8. alias\_map.insert(make\_pair(new\_name, alias\_end));
9. alias\_end += 128;
10. pthread\_mutex\_unlock(&alias\_end\_lock);
11. **else** // found -> update alias kvs file
12. pthread\_mutex\_lock(&alias\_busy);
13. pwrite(fd\_alias, file\_buf, 128, iter->second);
14. pthread\_mutex\_unlock(&alias\_busy);
15. **end**

Algorithm 5: handling PATCH command in the processing function

**2.2.3 name\_handler to support mapping alias to existing name**

Basically, the name\_handler will be given a search\_name and return success code and a result\_name which is the 40 HEX httpname. If no httpname indicated or there is an alias chain loop, the name\_handler will return a fail code.

**Copy search\_name into result\_name**

**Return 1**

**YES -> httpname**

**name\_handler()**

**search\_name is 40 HEX?**

**YES**

**Obtain existing\_name from KVS file**

**YES - httpname**

**Is name 40 HEX?**

**found in cache?**

**NO -> alias**

**Copy search\_name into result\_name**

**Return 1**

**NO -> alias**

**Reset search\_name with existing\_name**

**found in cache?**

**NO**

**…… Continue this loop**

**After 8 loops, there may be an alias chain -> return -1**

**……**

**Return -1**

In terms of the return value of name\_handler(), 1 indicates successful obtaining httpname, and -1 indicates no valid httpname is found. Beased on the return value, the caller is able to know the status of the result\_name, valid or not valid.

1. **Input :** char \* **search\_name** char \* **result\_name**
2. **Shared Variables :** map<string, uint32\_t> **alias\_map** int32\_t **fd\_alias**
3. **if** strlen(search\_name) == 40 **then**
4. memcpy(result\_name, search\_name, 40)
5. **return** 1
6. **end**
7. map<string, uint32\_t>::iterator iter
8. char temp[128]
9. **for** i = 1 -> 8 **then**
10. **if** (iter = alias\_map.find(search\_name)) == alias\_map.end() // not found
11. **return** -1
12. **else** // found
13. pread(fd\_alias, temp, 128, iter->second)
14. char \* ptr = temp + strlen(temp) + 1
15. memset(search\_name, 0, 128)
16. strcpy(search\_name, ptr) // obtain existing\_name
17. **if** strlen(search\_name) == 40 **then**
18. strcpy(result\_name, search\_name)
19. **return** 1
20. **end**
21. **end**
22. **end**

Algorithm 6: name\_handler() obtain httpname by given search\_name

**2.2.4 changes in PUT and GET execution**

Call function name\_handler() before every execution start. If the function returns -1, tehn send 404 Not Found response back to client; otherwise, the program has 40 HEX httpname to use.

**Relevant Variables :** char \* **new\_name** char \* **httpname**

1. **if** name\_handler(new\_name, httpname) < 0 **then**
2. send(cl, "HTTP/1.1 404 Not Found\r\n", 24, 0)
3. **else**
4. …… other execution in GET and PUT

Algorithm 7: call name\_handler() before regular execution in GET and PUT

**2.3 Multithreading and concurrency control**

Because this design include writing into a file, two new shared variables which are mutex alias\_end\_lock and alias\_busy are introduced to control only one thread can write into the file and only one thread can access the end of alias KVS file to insert a new entry simultaneously.

There is no risk for multiple threads read an entry in alias KVS file since we don’t need to control the consistency.