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
recv(cl, buf, sizeof(buf), 0)
ptr = strstr(buf, " ")
strncpy(action, buf, ptr - buf)  // obtain action code
if strcmp(action, "PATCH") == 0 then  // PATCH command
    ptr_tem = strstr(ptr + 1, " ")
    if buf[ptr + 1 - buf] == '/' then  // delete '/' char
        ptr += 1;
    end
    memcpy(existing_name, ptr + 1, ptr_tem - (ptr + 1))  // obtain existing_name
    ptr = strstr(ptr_tem + 1, "\r\n\r\n")
    if buf[ptr_tem + 1 - buf] == '/' then  // delete '/' char
        ptr_tem += 1
    end
    memcpy(new_name, ptr_tem + 1, ptr - (ptr_tem + 1))  // obtain new_name
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.

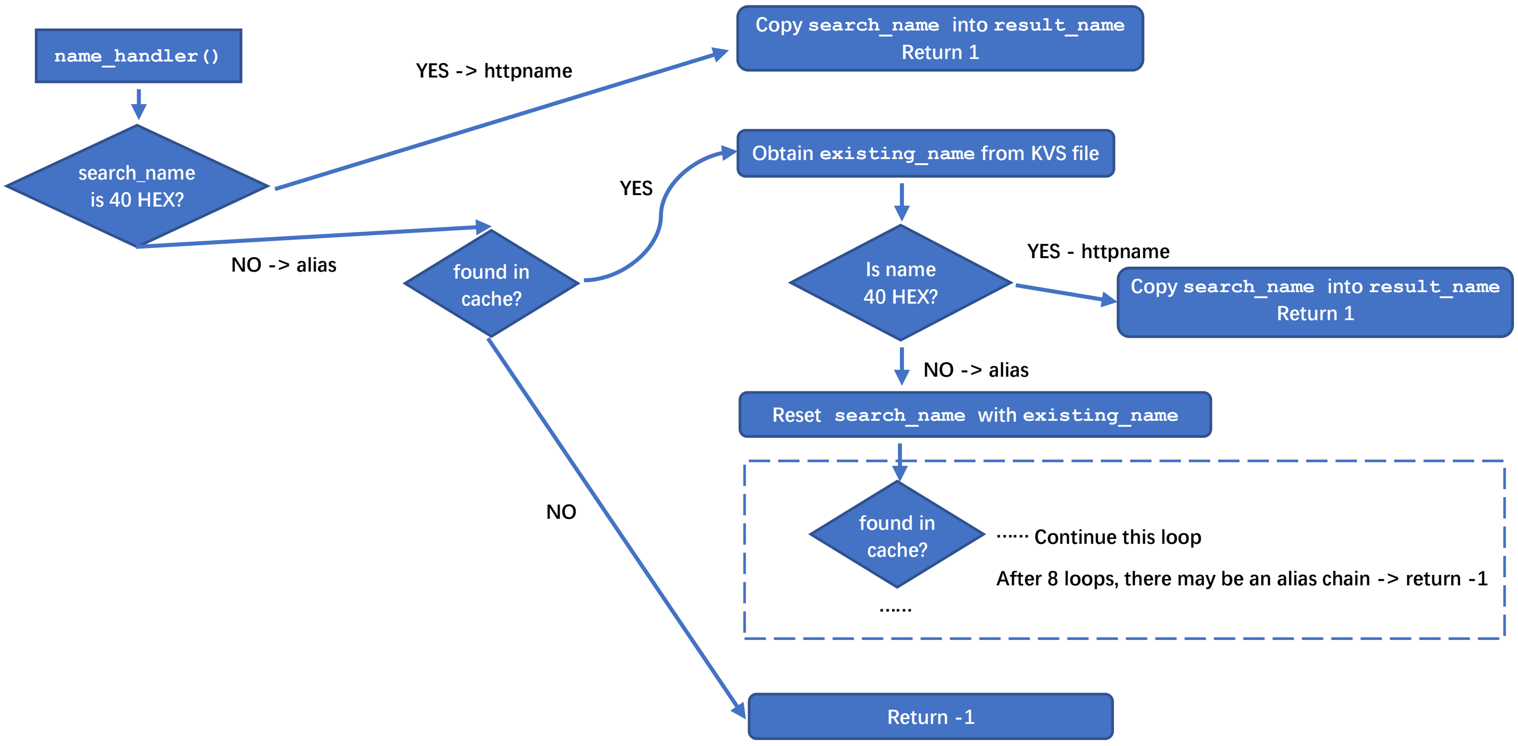
```
Relevant Variables  :  map<string, uint32_t> alias_map
Shared Variables    :  int32_t fd_alias uint32_t alias_end mutex alias_end_lock
                       mutex alias_busy
map<string, uint32_t>::iterator iter;
memcpy(file_buf, new_name, strlen(new_name));
memcpy(file_buf+strlen(new_name)+1, existing_name, strlen(existing_name));
if (iter = alias_map.find(new_name)) == alias_map.end() then  // not found
    // store alias in KVS file and update cache
    pthread_mutex_lock(&alias_end_lock);
    pwrite(fd_alias, file_buf, 128, alias_end);
    alias_map.insert(make_pair(new_name, alias_end));
    alias_end += 128;
    pthread_mutex_unlock(&alias_end_lock);
```

```
else  // found -> update alias kvs file
    pthread_mutex_lock(&alias_busy);
    pwrite(fd_alias, file_buf, 128, iter->second);
    pthread_mutex_unlock(&alias_busy);
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.


name_handler()
search_name is 40 HEX?
YES -> httpname
Copy search_name into result_name Return 1
NO -> alias
found in cache?
YES
Obtain existing_name from KVS file
Is name 40 HEX?
YES - httpname
Copy search_name into result_name Return 1
NO -> alias
Reset search_name with existing_name
found in cache?
...... Continue this loop
After 8 loops, there may be an alias chain -> return -1
......
NO
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.

```
Input            : char * search_name  char * result_name
Shared Variables : map<string, uint32_t> alias_map  int32_t fd_alias
if strlen(search_name) == 40 then
   memcpy(result_name, search_name, 40)
   return 1
end
map<string, uint32_t>::iterator iter
char temp[128]
for i = 1 -> 8 then
   if (iter = alias_map.find(search_name)) == alias_map.end()  // not found
      return -1
   else  // found
      pread(fd_alias, temp, 128, iter->second)
      char * ptr = temp + strlen(temp) + 1
      memset(search_name, 0, 128)
      strcpy(search_name, ptr)  // obtain existing_name
      if strlen(search_name) == 40 then
         strcpy(result_name, search_name)
         return 1
      end
   end
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
if name_handler(new_name, httpname) < 0 then
   send(cl, "HTTP/1.1 404 Not Found\r\n", 24, 0)
else
   ...... 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.