Evelyn Chen 704332587
Yanzun Huang 304170110
Bob Wu 604450152

CS 130 Assignment 9
Feature of Your Choice, Final Presentation: Design Document

I.Description of the Webserver

Our webserver processes requests via HTTP, and it consists of different handlers, such as echo handler, static handler, status handler, and not found handler.

- Original Handlers
  - Echo handler
    - Echo handler echos the request back to the server.
  - Static handler
    - Static handler returns a specified file to the server.
  - Status handler
    - Status handler displays status of the handler the number of requests server received or the kind of request handlers that exist.
  - Not found handler
    - Not found handler returns a 404 not found response.
- New handlers added for new features
  - ConfigTestingHandler
    - Config testing handler changes the config file from “config” to “config_demo” to include a new StaticHandler with static3 as prefix to test for server reconfigurability feature
  - ConfigResetHandler
    - Config reset handler changes the config back from “config_demo” to “config”
  - CloudHandler
    - Cloud handler serves files from a cloud storage service like Amazon S3
- Deployment
  - Our webserver is deployed to AWS so anyone on the internet can access it.
  - The public IP address of our webserver is http://54.201.90.157/

II.Alternatives Considered For Features

We have considered implementing http compression. However, after looking through the documentations of compression, we decided it will be our best interest to allocate our time to do many smaller features instead of one big feature. By implementing many small features, we get to learn different possibilities we can add to a webserver.

III. Why We Chose The Features

We decided to implement three features, which are markdown, reconfigurability of server, and file serving from cloud storage device. We implemented markdown because markdown library is readily available, and we wanted to learn to implement code using a library. We implemented reconfigurability of server because we think this capability is important. By implementing this, we do not have to rerun the server after changing the config file in the middle of server running. Lastly, we implemented file serving from cloud storage device because we want to explore more on AWS-ec2.

IV. Implementations of The Features

1. Implement Markdown rendering
   a. For the markdown feature, we checked the recommended markdown libraries in this link: http://stackoverflow.com/questions/889434/markdown-implementations-for-c-c
   b. We then used the library from cpp-markdown licensed under MIT: https://sourceforge.net/projects/cpp-markdown/
   c. Followed the readme of the library
   d. After adding the markdown files from the cpp-markdown library to our repository
      i. added “Boost::Regex” library to our Makefile
      ii. Added “#include “markdown.h”” inside static handler file
      iii. In static handler file, created a markdown::Document object, fed in some input with string through the read function
      iv. Called the write function to write out the HTML code to an ostream

2. Make the server reconfigurable without restarting it
   a. Inside webserver doAccept function, config file is parsed again to reconfigure the parameters if the config file is changed while server is running
   b. Config file is parsed every time we get a new request
   c. When the config file is changed, webserver is reconfigured without the need to run again
   d. Created a new config file called “config_demo”
      i. Copy the same content from original config
      ii. Then, add a new StaticHandler defined in addition to what original config has:
```
path /static3 StaticHandler {
        root ./test_folder;
}
```
   e. We also added two handlers for testing purpose
   f. Added these two lines to config
      i. path /changeConfig ConfigTestingHandler {}
      ii. path /resetConfig ConfigResetHandler {}
   g. ConfigTestingHanlder
      i. Change the config file from “config” to “config_demo”
   h. ConfigResetHandler
      i. Change the config back from “config_demo” to “config”

3. Serving files from a cloud storage device, Amazon S3
   -Implement file serving from cloud storage device. When you know the path of a public file inside the cloud storage service, you should be able to access it from the webserver.
   -Files can be added at anytime to the cloud storage device and the user would still be able to access it within the already running webserver.
   a. The CloudHandler could be specified in the configuration file in the following format:
      ```
      path /cloud CloudHandler {
            root s3-us-west-1.amazonaws.com;
        }
      ```
      -The url provided as a value for root is the DNS of the Amazon S3 service that contains the location of the bucket containing our files.
      -The /cloud path is mapped to the CloudHandler.
   b. The webserver takes in the url: http://54.201.90.157/cloud/cs130-assign9/"file name".
   c. The CloudHandler has the same basic functionalities as a static file handler, with the only difference being that the cloud file is available through a url.
      i. Appends the content of the file as the body of the response.
      ii. Files are available to be served if they are made public on the cloud storage.
   d. The CloudHandler also possess some features of the proxy handler, as it sends a request to the cloud storage for the requested object.
      i. A connection is made between webserver and the cloud storage service by querying the DNS of the service, s3-us-west-1.amazonaws.com.
      ii. The handler will send a request for the following object path parsed from the url.
         1. /cs130-assign9/"file name"
      iii. It will then receive a response from the cloud storage. The response is parsed into:
         1. Response Status (HTTP/1.1 200 OK)
         2. Response Headers (header:value)
         3. Response Body (content of the file if it exists).
   e. The above parts are then appended to the response for the webserver
      i. First it will set the status through Response::SetStatus.
      ii. Then it will set all the headers through Response::SetHeader.
      iii. Then it will set the body by appending the content of the file through Response::SetBody.

V. How to Demo The Features

1. Markdown
   a. http://54.201.90.157/static/markdown_sample.md
   b. The markdown file will display in html format
   c. When you right click for "inspect element", the type will show "html"

    d. To test for other markdown files, one can create a markdown file, put it inside test_folder, and use localhost:<port_num>/<static handler prefix>/<markdown_file> to test it. <port_num> and <static handler prefix> depend on the specifications in the config file

2. Make the server reconfigurable
    a. In this part of the demo, we want to show that the webserver is reconfigurable without rerunning it. We have a ConfigTestingHandler with prefix changConfig and ConfigResetHandler with prefix resetConfig that are used to change the config to config_demo, which adds a new static handler with static3 as prefix. Originally, static3 will return an error message, but after calling changeConfig handler (which switch the config to config_demo), static3 will be mapped to test_folder, and we can access the files inside test_folder with the URL.
    a. http://54.201.90.157/static3/
        i. It will return an error message because static3 is not specified in config file
    b. http://54.201.90.157/changeConfig
        i. Let the webserver change to another config file config_demo instead of config
        ii. server->setConfigFileName("config_demo");
        iii. In config_demo, it has an new line of static handler:
```
path /static3 StaticHandler {
        root ./test_folder;
}
```
    c. http://54.201.90.157/static3/apple.jpg
        i. Can access files from test_folder using static3 now
    d. http://54.201.90.157/resetConfig
        i. Reset back to config file from config_demo file

3. Serving files from a cloud storage device, Amazon S3.
    a. http://54.201.90.157/cloud/cs130-assign9/”file name”
    b. Several different file types are stored in the Amazon cloud storage service, S3, for demo purposes.
    c. For the purpose of this demo, a premade bucket with existing files is already created.
    d. Can show existence of image through following links:
        i. http://s3-us-west-1.amazonaws.com/cs130-assign9/Fish.jpg (made public)
        ii. http://s3-us-west-1.amazonaws.com/cs130-assign9/s3.txt (made public)
        iii. http://s3-us-west-1.amazonaws.com/cs130-assign9/Pond.JPG (private)
    e. The webserver can serve the above files in the following url format:
        i. http://54.201.90.157/cloud/cs130-assign9/”file name”
        ii. The “file name” for the above 3 links would be Fish.jpg, s3.txt, Pond.JPG.
    f. If the instructors would like for further demonstration:
        i. Can ask student to make Pond.JPG public, and then try to access url for image again.
        ii. Could also ask student to add new files to the cloud storage, and see if files are accessible after student makes them public.
    g. The config file format for adding this new handler should be:

```
path /cloud CloudHandler {
    root s3-us-west-1.amazonaws.com;
}
```

The string after root should represent the DNS of the S3 service.

h. Multiple CloudHandlers can be created, and each can have its own unique DNS of the S3 service to serve from.
i. As seen from the first line in this section, after entering the path "/cloud", the bucket path is needed. In this case, the bucket path is "/cs130-assign9".
j. After the bucket path, a file path is needed. In this case, the file path is " /(any file name currently in bucket that is made public)".
k. The CloudHandler also handles 404, 403 requests. This is purely a design choice as Amazon S3 returns the 404, 403 status code along with a xml file that shows the file requested is not there or forbidden, respectively.