CS 118 Computer Network Fundamentals

Project 1: Concurrent Web Server using BSD Sockets

Team Member 1

Name: Darin Minamoto

Student ID: 704140102

SEASnet login name: darin

Team Member 2

Name: Lingyu Zhang

Student ID: 404205755

SEASnet login name: lingyu

1. Design Description

Our server reads requests from the client, parses the request string to find the path name of the file being requested, and returns a HTTP response with the file appended after the headers. Most of our code is inside the header file parse.h:

Main functions in parse.h:

char* parseRequestMessage(char* request, long* size)

char* getContentTypeFromPath(char *path)

char* appendHeaderToResponse(char* content, char* contenttype, long *content_size)

Brief descriptions of the main functions in parse.h:

char* parseRequestMessage(char* request, long* size)

This function is the main function that parses the request message sent from the client, finds the requested file, and generates a HTTP response. This function also checks for incorrectly formatted requests and protects against directory traversal attacks (such as trying to access files outside the server's directory using paths like /, ../, or ~/). If the request is invalid or a directory traversal attack, the function will return NULL.

char* getContentTypeFromPath(char *path)

This function is called by parseRequestMessage to get the content-type header field for the specified path. It does this by finding the extension of the file and comparing it to supported extension types. If the extension is not supported, the function will return NULL.

char *appendHeaderToResponse(char* content, char* contenttype, long *content_size)

This function is called by parseRequestMessage to append the response header to the content requested by the client. If the content type is not supported (right now only content types starting with "text", "image", and video are supported), then the function returns NULL.

2. Problems and Solutions

We initially wrote the binary without the header for images and videos because it worked in Chrome but when we tested it on Firefox
it tried to parse the binary and crashed. We solved this by appending the header to the content for images and videos.

3. How to Compile and Run

Compile:

```
   $ make
# if make fails because timestamp is messed up
   $ touch –a –m –t 201405040000.00 ./*
   $ make
```

Run:

```
   $ ./serverFork <port number> # start server on specified port
```

4. Sample Outputs and Explanation

To test our server, first, we start the server on port 8081:

```
   $ ./serverFork 8081
```

Then, we connect to our server from Firefox with the URL:

```
   http://localhost:8081/<object-requested>
```

We tested our server in the CS118 Ubuntu distribution and include the results and explanation below.

a) Requesting an html file (index.html)

We included in the directory of the server an html file `index.html`

```
   <html>
          <body>
                <h1>Hello, world!</h1>
                <img src="chipmunk.gif">
                <img src="finalscore.jpg">
         </body>
   </html>
```

requesting `chipmunk.gif` and `finalscore.jpg` in the current directory.

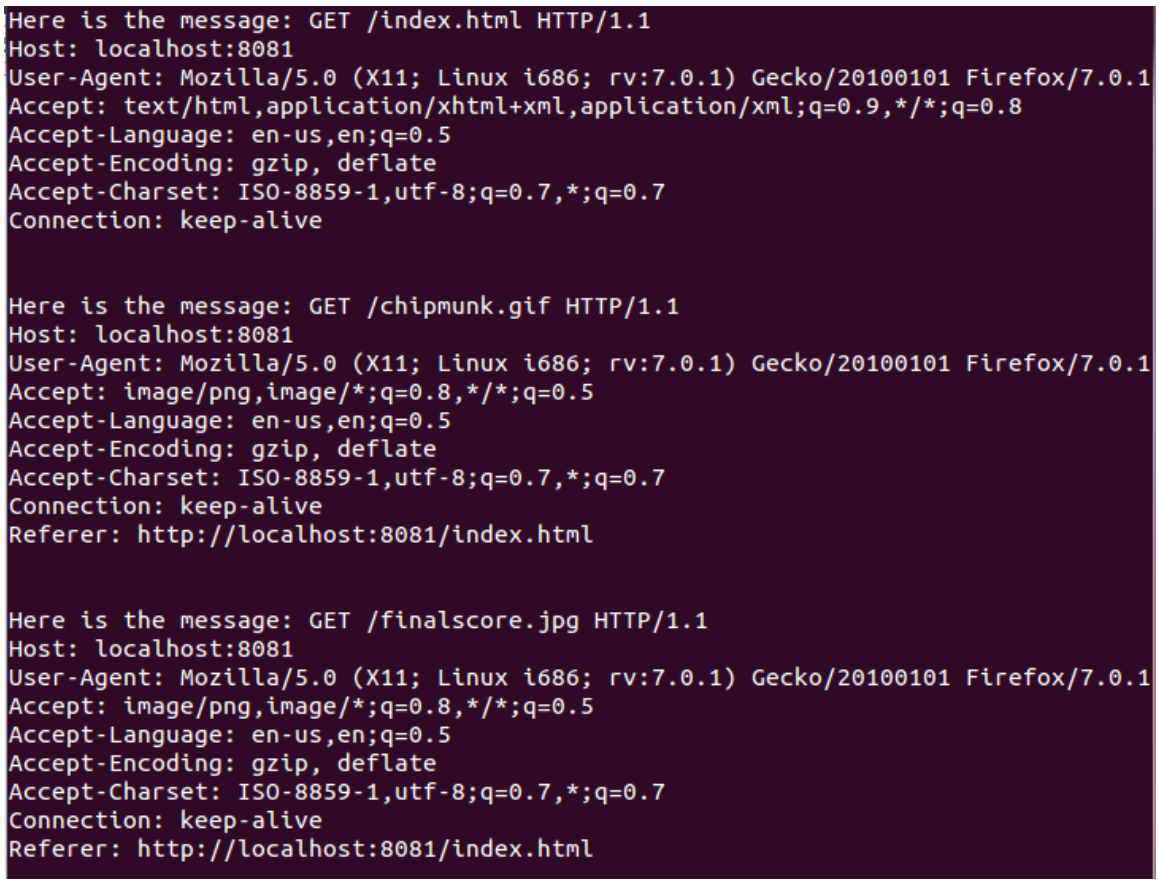

Figure 1

Figure 1 is a screenshot of the console with dumped HTTP request messages of index.html. We can see that the browser sends three requests: the base html file (index.html) and the two referenced objects (chipmunk.gif and finalscore.jpg). We explain the request message for chipmunk.gif as an example.

- The Request-Line consists of three fields: `GET` (method), `/chipmunk.gif` (URL of requested object), and `HTTP/1.1` (HTTP version).
- The Host request-header field specifies the Internet host and port number. Since we are running the server and the client on the same machine, the hostname is `localhost`.
- The User-Agent request-header field contains information about the user agent initiating the request. In our case this is `Firefox`.
- The four subsequent request-header fields starting with Accept specify the format information acceptable for the response (media types, languages, encoding, and character sets).
- The Connection request-header field specifies that the client support persistent HTTP connection.

- Finally, the Referer request-header field identifies the URI of the page that linked to the resource being requested. In our case this is the index.html file.

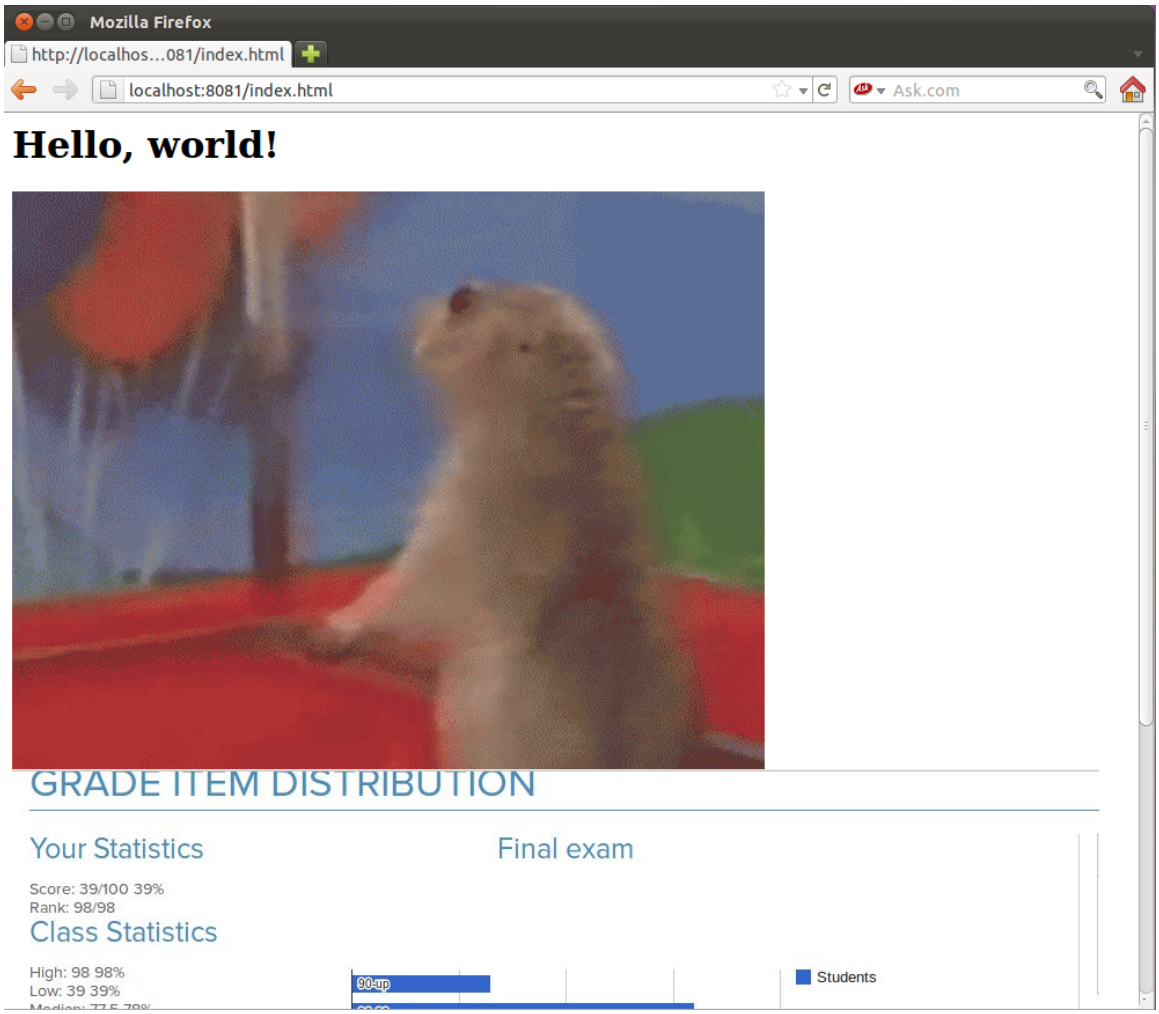

Figure 2

Figure 2 is a screenshot of the Firefox browser when we request the html file `index.html` from `localhost` on port `8081`. The `"Hello, world!"` message, the `chipmunk.gif`, and the `finalscore.jpg` are properly displayed.

b) Requested object not found

If the client requests an object that is not in the server's directory, the server responds with an error message. Figure 3 and 4 below are the results of requesting

`http://localhost:8081/blah`

```
Here is the message: GET /blah HTTP/1.1
Host: localhost:8081
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:7.0.1) Gecko/20100101 Firefox/7.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
```

Figure 3

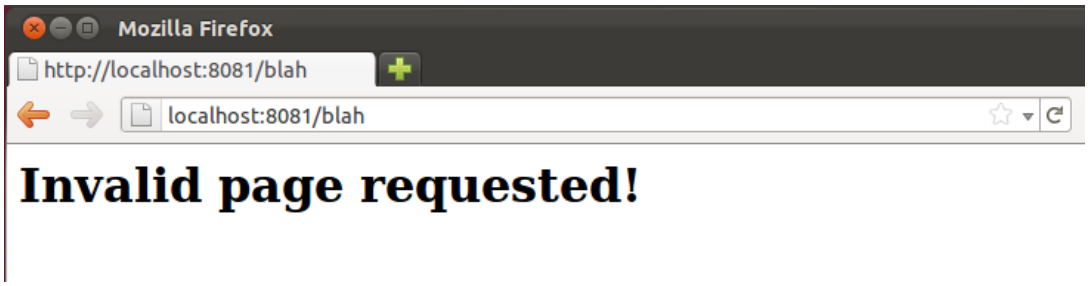

Figure 4

3) Support for video

We added support for .mkv files. However, since we were unable to configure the Firefox plug-in, the bowser cannot stream the requested video. Instead, it prompts an option for download. The results are shown in Figure 5 and 6 below.

```
Here is the message: GET /dango.mkv HTTP/1.1
Host: localhost:8081
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:7.0.1) Gecko/20100101 Firefox/7.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
```

Figure 5

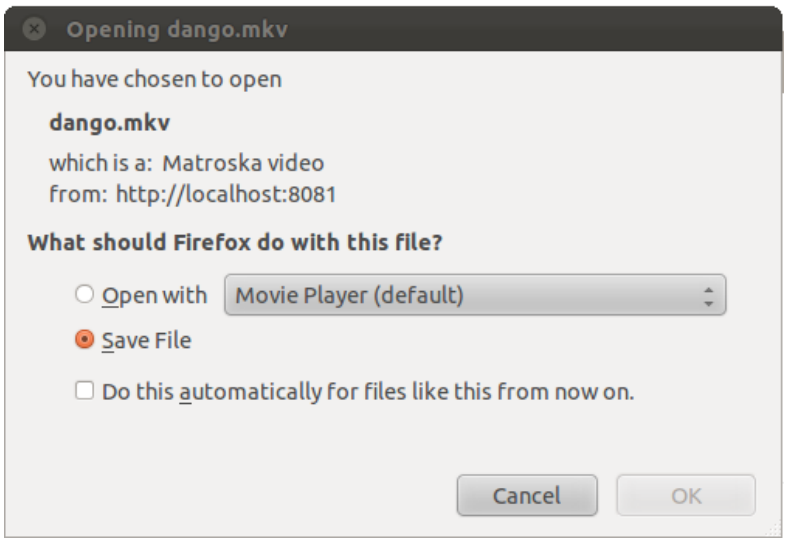

Figure 6