Alan Kha 904030522
Rukan Shao 504355612

CS 118 Lab 1

1 PART A

```
1  Here is the message: GET /helloworld HTTP/1.1
2  Host: 127.0.0.1:1025
3  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:20.0)
   Gecko/20100101 Firefox/20.0
4  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: g
```

The first line (aka the request line) consists of the Method, URL, and HTTP version field. Our method is `GET`, which is when our browser requests an object. Our URL is `/helloworld`, which identifies the object we are requesting, and our HTTP field is `HTTP/1.1` which specifies that we are using HTTP version 1.1.

The remaining 5 lines are header lines. The "Host:" line specifies the host on which the object resides. The "User-Agent:" line specifies the browser that is making the request to the server. The "Accept:" line indicates the list of media ranges that are acceptable. The q's can be used to indicate the priority of less-desirable media types.

In our case, `text/html` and `application/xhtml+xml` do not have a q value, meaning they are the two preferred media types. If they are not available, the server will send `application/xml` has q=0.9, which is greater than the alternative. So the server will send `application/xml` if it is available, but `text/html` and `application/xhtml+xml` are not available. If none of those are available, then the server will send `*/*` which has q=0.8. `*/*` means any media type, with any subtype. In other words, if none of the 3 more desirable types are available, the server will just send whatever it has.

The "Accept-Language" line works similarly to the accept line, but it indicates the desired natural language, as opposed to media type. In our case, we desire US English as our language type. If that is not available, we will accept English as the language type.

The "Accept-Encoding" line restricts the content coding values in the server's response. For example, the server may want to use gzip or deflate compression when sending its response, in order to save bandwidth.

2 PART B

2.1 DESCRIPTION

The server receives a message from the browser and stores it in the buffer and extracts the filename from the message. Once it has the filename, the server checks if the file is available in our folder. If the file is not found, a 404 error is sent to console and written to the socket before the process exits. Otherwise, the server opens the file, loads it into memory, and writes it to the socket.

2.2 CHALLENGES

The first issue we ran into was an error where requested images would not display at all. Instead, the browser would return an error stating that the data in the image file could not be recognized.

The second issue we ran into one issue where small image files would load fine, but large ones, such as the flower image in the attached sample output page, would only load part of the way (25% or so). But the issue was resolved by extending the buffer.

2.3 INSTALLATION

1. Navigate into the SocketCS folder from the terminal
2. Run `make` to compile.
3. Start the server with `./serverFork ####` where `####` is the port number
4. Open a browser
5. Enter `http://127.0.0.1:####/filename` to access a file called `filename`.

2.4 SAMPLE OUTPUTS

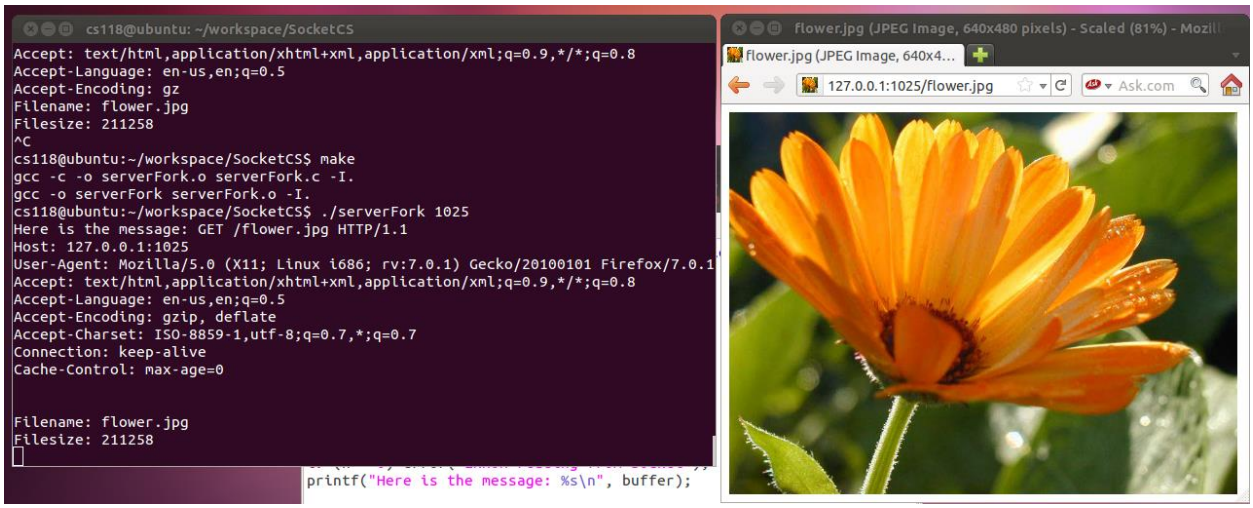

Figure 1. Loading flower.jpg

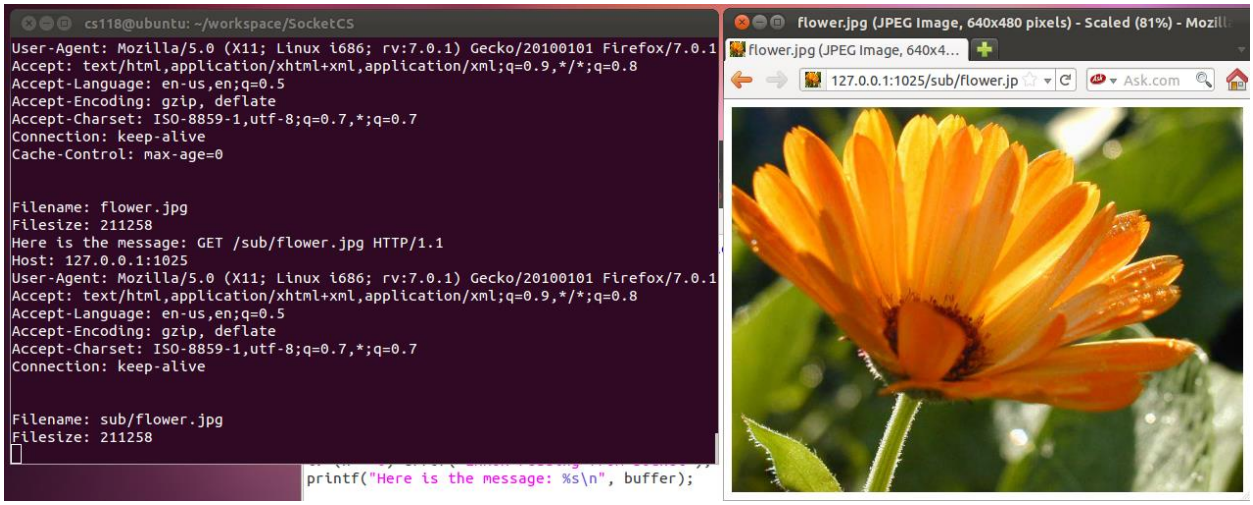

Figure 2. Loading sub/flower.jpg

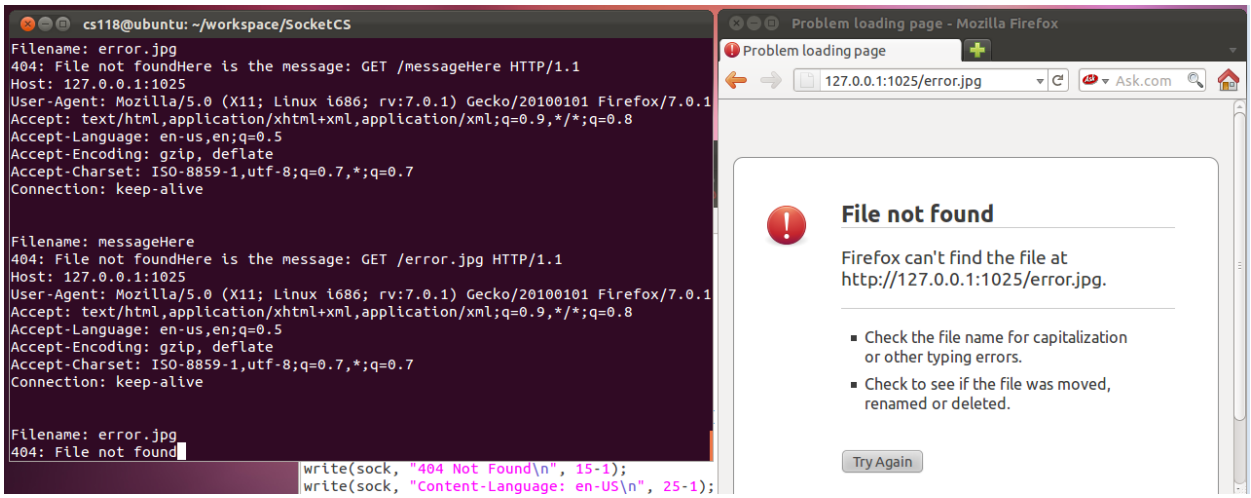

Figure 3. Loading invalid file

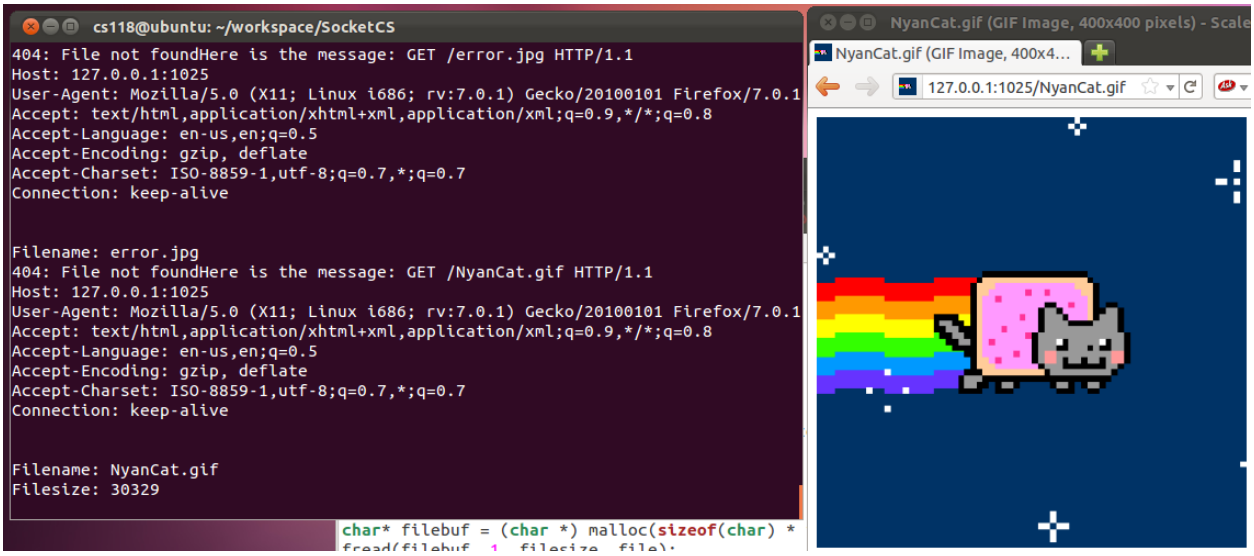

Figure 4. Loading an animated gif