# Design Document: HTTP server

Anthony Ho
CruzID: Ankho

## 1) Goals

The goal of this assignment is to implement a simple web server that handles GET and PUT requests. The server will store files in a directory on the server and be able to receive and send files.

## 2) Design

The design will include 3 parts. When the program initializes, we will begin by opening a socket and then proceed to parsing the header, and lastly, we will be handling GET vs PUT requests.

### 2.1) Opening a socket

Open a network socket by including and using the functions in sys/socket.h like so:

```
--------------------------------------------------------------------------------------------------------------
Input: Argument count: argc
Input: Array of arguments: argv

Initialize hostname and port as argv[1] and argv[2] (The command line arguments)
Initialize a main socket using socket()
Use setsockopt()
Use bind() to bind the socket to the hostname and port
Use listen() to put our socket into a passive state and wait for a client
while(1):
    new_socket = accept()
    Create a buffer of size 32KiB
    Use read() to save the header sent by the client
--------------------------------------------------------------------------------------------------------------
```

## 2.2) Parsing the Header

After receiving the header, we need to parse the header to be able to fulfill the request. We will be parsing the header by first isolating the first line of the header that contains the GET or PUT request and the 27 character resource name. This will be shown in the following algorithm:

```
-----------------------------------------------------------------------------------------------------------
Use strtok() to split the string using the delimiter of a space (first two tokens will be
request and the resource name)
Create a pointer to the “Content-Length:” header using strstr()
Create a variable to store the content length
If the content-length header exists:
    Add 16 to the pointer (to get to the number)
    while(digits remain):
        If current char is a digit:
            Add current digit to content length variable
            Increment pointer
Create variables for “GET” and “PUT”
Compare the first string after splitting with the variables to determine if its a GET or PUT
Pass the socket and the resource name to the respective function
-----------------------------------------------------------------------------------------------------------
```

## 2.3) GET vs PUT

Depending on the request type, the server will either send the contents from a file on the server to the client or store a file from the client onto the server. This will be handled by two different functions that are defined as so:

### 2.3.1) GET

The get function will take the 27 character resource name and find that file and send the contents to the client with a content-length and status code 200 OK. If the file does not exist on the server, it will return status code 404 file not found. This will also check the validity of the resource name. This will be done with the following algorithm:

```
-----------------------------------------------------------------------------------------------------------
Input: socket
Input: char* resName

Initialize an array that contains all the valid characters for a resName. (A-Z, 0-9, -, _)
Check if the resource name is the correct length (27) and contains proper chars
^ Else return status code 400 through the socket
```

```
Use open() to find the file on the server
if(file exists and cannot be accessed):
    Write status code 403 through the socket
Else
    Initialize a variable to track content length
    if(file exists) :
        Loop through file to get the length of the entire file
        Create a buffer of size 32KiB
        Write status code 200 through the socket
        Write content length through the socket
        While there is still content to be written:
            Read the file
            Write the content through the socket using write()
    Else if file doesn't exist:
        Write status code 404 through the socket
```

------------------------------------------------------------------------------------------------------------

## 2.3.2) PUT

The put function will start similarly to the get function as it will check the resource name for any errors. The function will then attempt to save the file from the client into the server. If a content length is provided, it will only store up to the content length, otherwise it will save the entire file. This function will send a status code 200 OK if it is able to save the content into a file on the server. It will send a status code 201 File Created if the file did not previously exist and it creates the file.

------------------------------------------------------------------------------------------------------------

```
Input: socket
Input: char* resName
Input: int contentLength
Initialize an array that contains all the valid characters for a resName. (A-Z, 0-9, -, _)
Check if the resource name is the correct length (27) and contains proper chars
^ Else return status code 400 through the socket
if(file exists and cannot be accessed):
    Write status code 403 through the socket
Use open() to check if the file already exists in the server
If the file does not exist:
    Use Open() to create a file with read and write permissions
    If a content length was provided:
        While content length isn’t reached:
            Read() content from the socket
            Write() the content to the file you created
            Update content length with new value
    Else
        While client doesn’t terminate connection:
            Read content from the socket
            Write the content to the file you created
            If client terminates connection:
                stop
    Write status code 200 through the socket
Else (file already exists)
    Use open() to overwrite the file
    If a content length was provided:
        While content length isn’t reached:
            Read() content from the socket
            Write() the content to the file you created
            Update content length with new value
    Else
        While client doesn’t terminate connection:
            Read content from the socket
            Write the content to the file you created
            If client terminates connection:
    Write status code 201 through the socket
```