Design Document: httpserver

1. Goals

The goal of this program is to create a server that responds to GET and PUT commands. The files are printed using non FILE * functions. The program will listen to a user-specified port and respond to PUT and GET using HTTP style headers.

In order to comply with curl behavior during testing, GET headers will always include a "Content-Length" line even if it is zero.

2. Design

The design is separated into three parts. The program first initialize the server using arguments. Then the server waits and accepts a connection. Finally, the server responds to a request accordingly with a http header.

2.1 Handling Arguments

The first argument to *httpserver* is the address that maybe a hostname or IP address. The second argument is the optional port number, port 80 by default. In case the address given is "localhost" or a hostname, gethostname() is used to get the hostname of the server. The program then strcmp the input with the hostname returned from gethostname(). Otherwise assume the format is correct and place into struct sockaddr_in. If any error occur, it would be handled by bind. Arguments handling is shown in Algorithm 1.

```
Input: Argument count: arg_count
Input: Argument address: arg_add
Input: Argument port number: arg_port
Output: Address type: add_typ
Output: Address: address
Output: Address struct: addr
s_fd = socket(AF_INET, SOCK_STREAM, 0);
if s_fd == -1 then
|   exit
end
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_adr = INADDR_ANY;
addr.sin_port = htons(80);
if gethostname(hostname, len) == -1 then
|   exit
end
if  strcmp(arg_add, hostname) == 0 || strcmp(arg_add, "localhost") == 0 then
|   addr.sin_addr.s_addr = inet_addr(hostname);
else
|   addr.sin_addr.s_addr = inet_addr(arg_add);
end
```

```
if arg_count == 3 then
|   addr.sin_port = htons(arg_port);
end
if bind(s_fd, (struct sockaddr_in*)&addr, sizeof(addr)) == -1 then
|   exit
end
```

Algorithm 1. Handling Arguments

2.2 Listening and Accepting

Now that the address and port are put in struct sockaddr_in, the struct instance is passed to bind(). Then listen() waits for a connection from a client.

```
if listen(s_fd, 3)  ==  -1  then
|   exit
end
acc_soc = accept(s_fd, (struct sockaddr_in*)&addr, sizeof(addr), (socklen_t*)&addrlen);
if acc_soc == -1 then
|   exit
end
```

Algorithm 2. Listen and Accepting

2.3 Creating Http Header

```
read( acc_soc, buffer, 1024);
sscanf(buffer, “%s %s”, command, filename);
if strcmp(command, “GET”) then
|   write()
else if strcmp(command, “PUT”) then
|   fetchFile(filename)
end
send(acc_soc , response, strlen(response) , 0 );
```

Algorithm 3. Creating Header

2.4 FetchFile