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.

Error handling is done with an if statement and err(1, "function() failed"), not included in pseudocode.

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 = htons(atoi(arg_add);
end
if arg_count == 3 then
|    addr.sin_port = htons(atoi(arg_port));
end
if bind(s_fd, (struct sockaddr*)&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
while true
|    if acc_soc = accept(s_fd, (struct sockaddr_in*)&addr, sizeof(addr),
(socklen_t*)&addrlen) == -1 then
|    |    exit
|    end
|    handle_client (acc_soc)
end
```

Algorithm 2. Listen and Accepting

2.3 handle_client()

Inside the while loop with accept, handle_client reads the message are identify the request and filename. A response is made using concat(). If the request is PUT, a file is made using write() with the filesize of content-length and data from the received header. If the request is GET, read() tries to find the file with the same name. If the file exists, the content is copied into a buffer. strcat() concatenate the buffer into the response. Finally, the response is sent using send().

```
Input accepted socket: acc_soc
read( acc_soc, buffer, sizeof(buffer));
sscanf(buffer, "%s %s %*s %*s %*s %*s %*s %*s %*s %d %s", command,
     filename, &size, data);
if strcmp(command, "PUT") == 0 then
|    fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR);
|    if fd == ERR then
|    |    strcat(header, "400 bad request\r\n");
|    else
|    |    write(fd, data, sizeof(data))
|    |    strcat(header, "201 Created\r\n");
|    end
else if strcmp(command, “GET”) == 0 then
|    fd = open(filename, O_RDONLY);
|    if (fd == -1) then
|    |    strcat(header, “400 bad request\r\n”)
|    else
|    |    strcat(header, “200 ok\r\n”)
|    |    read(fd, data, sizeof(data)
|    |    close(fd)
|    |    sprintf(buffer, "Content-Length: %d\r\n%s\r\n", sizeof(data), data);
|    |    strcat((char *)header, (char *)buffer);
|    end
|    strcat((char *)header, "500 Internal Server Error\r\n");
end
strcpy(response, (char*)header);
send(soc_fd, (char*)header, HEADERMAX, 0);
```

Algorithm 3. handle_client()