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").

2. Design

The design is separated into 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. getaddrinfo() is used to get the information needed into struct addrs. Members of addrs is then passed to socket and bind. Each networking function has their own error handling if statement. 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
if  argc < 2 or argc >= 4 then
|     err(1, "missing argument(s)\n argc: %d", argc);
end
struct addrinfo *addrs, hints = {};
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
if arg_count == 3 then
|    getaddrinfo(argv[1], argv[2], &hints, &addrs);
else
|    getaddrinfo(argv[1], "80", &hints, &addrs);
end
s_fd = socket(AF_INET, SOCK_STREAM, 0);
if s_fd == -1 then
|    err(1, message)
end
```

```
if setsockopt(main_socket, SOL_SOCKET, SO_REUSEADDR, &enable,
        sizeof(enable)) == -1 then
|   err(1, message)
end
if bind(main_socket, addrs->ai_addr, addrs->ai_addrlen) == -1 then
|   err(1, message)
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
|   err(1, message)
end
while true
|   if acc_soc = accept(s_fd, (struct sockaddr_in*)&addr, sizeof(addr),
(socklen_t*)&addrlen) == -1 then
|   |   err(1, message)
|   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(). sscanf() detects the request in buffer.

The first line of the header is read to get the request. Using strstr(), the pointer to the beginning and end of line “content-length” is found. The size of the content is saved and converted with atoi().

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().

Finally, a while loop is used to read and send content requested if necessary. read() and send() would use the same buffer and size to be read. A counter decreases to keep track of the data remaining to be read and send. When complete, close() is used to close the file descriptor.

```
Input accepted socket: acc_soc
read( acc_soc, buffer, sizeof(buffer));
```

```
sscanf(buffer, "%s %s", command, filename, size, data);
read(acc_soc, buffer, sizeof(buffer));
substring_start = strstr(buffer, "Content-Length: ");
if (substring_start != nullptr then
|   substring_end = strstr(substring_start, "\r");
|   sub_len = substring_end - substring_start - 16;
|   strncpy(cont_len_substr, substring_start + 16, sub_len);
|   size = atoi(cont_len_substr);
|   if size > 0 then
|   |   read(soc_fd, (char *)payload, sizeof(payload));
|   end
 end
strcpy(header, “HTTP/1.1”);
if filename = “/” or filename size not 27 then
|   strcat(header, 403 forbidden\r\nContent-Length: 0\r\n”);
else if strcmp(command, "PUT”) == 0 then
|   if access(filename, W_OK) == 0 then
|   |   remove(filename)
|   end
|   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”)
|   |   fileSize = lseek(fd, 0, SEEK_END)
|   |   lseek(fd, 0, 0);
|   |   char fileData[fileSize]
|   |   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, headersize, 0);
if (payloadSize > 0) then
|   read(fd, payload, BUFMAX);
|   send(soc_fd, payload, BUFMAX, 0)
|   close(fd);
```

```
|    payloadSize = payloadSize - BUFMAX;
end
```

Algorithm 3. handle_client()