Design Document: HTTP server

Julia Sales
CruzID: jesales

1 Goals

The goal of this program is to implement a HTTP server that will respond to GET and PUT commands. These commands will correlate with read and write files. The HTTP server will store files in a directory persistently. Our program receives and sends files via http. The server will respond back with messages like 200 and 404.

2 Design

There are five parts to this design. First, we have to create a socket, which will take the address and set up the server and client. We then need to set up GET and PUT commands. After that we need to code accordingly to which status code will appear based on the conditions. We also need to figure out how to deal with invalid file names and file names that are valid but not not exist. We need to figure out the errors for these.

2.1 Setting up the Socket

Creating a sockaddr_in

```
1. struct hostent *hent = gethostbyname(SERVER_NAME_STRING /* eg “localhost” */);
2. struct sockaddr_in addr;
3. memcpy(&addr.sin_addr.s_addr, hent->h_addr, hent->h_length);
4. addr.sin_port = htons(PORT_NUMBER);
5. addr.sin_family = AF_INET;
```

Creating a socket

```
1. int sock = socket(AF_INET, SOCK_STREAM, 0);
```

Socket Setup for Server

Note: “cl” is now another socket that you can call read/recv and write/send on to communicate with the client

```
1. int enable = 1;
2. setsockopt(sock, SOL_SOCKET, SO_REUSEADOR, &enable, sizeof(enable));
3. bind(sock, (struct sockaddr *)&addr, sizeof(addr));
4. listen(sock, 0);
5. int cl = accept(sock, NULL, NULL);
```

Socket Setup for Client

Note: “sock” is now connected to the server. You can use read/recv and write/send on it to communicate with the server.

```
1. connect(sock, (struct sockaddr *)&addr, sizeof(addr));
```

2.2 PUT

```
1.  if arg[0] == “PUT”
2.      | output header
3.      | content length
```

2.3 GET

```
1.  if arg[0] == “GET”
2.      | output header
```