# **Jednoduchý web server, podporujúci podmnožinu protokolu HTTP/1.0.**

Server musí implementovať minimálne metódu GET. Požiadavky môžu prichádzať z rôznych WEB klientov, server preto musí podporovať obsluhu viacerých klientov súčasne.

GET <specifikacia\_suboru> HTTP/1.0\n\n

Ak je špecifikácia súboru zadaný iba znak lomky, potom server odošle súbor index.html, ak taký existuje v koreňovom adresári servera.

Špecifikácia súboru má tvar:

/<adresar>/<adresar>/…/<subor>

Pri spustení servera sa na príkazovom riadku špecifikuje port, na ktorom bude server prijímať požiadavky, ako aj adresár slúžiaci ako koreňový adresár serveru. Server nesmie umožniť prístup mimo podstromu, koreňového adresára (napr. Pomocou cesty obsahujúcej /../).

Funkčnosť servera bude overená pomocou webového prehliadača.

Implementácia servera bude pozostávať z dvoch častí:

1) Implementácia programu, ktorý bude funkčnosťou ekvivalentnýprogramu *netcat* (do miery vyžadovanej pre implementáciu servera) v jazyku C s použitím socketov

2) Implementácia samotného servera s využitím programu z bodu 1) Táto časť bude implementovaná v Bash-i a bude zabezpečovať spustenie servera, cyklickú obsluhu prichádzajúcich požiadaviek (parsovanie, ošetrenie prípadných chýb, nájdenie požadovaného, dokumentu a jeho odoslanie klientov) a korektné ukončenie servera.

Know how:

<http://tutorials.jenkov.com/java-multithreaded-servers/multithreaded-server.html> in Java

1. Start a ServerSocket listening (probably on port 80).
2. Once you get a connection request, accept and pass to another thread/process (this leaves your ServerSocket available to keep listening and accept other connections).
3. Parse the request text (specifically, the headers where you will see if it is a GET or POST, and the parameters passed.
4. Answer with your own headers (Content-Type, etc.) and the HTML.

**In C**

**Best tutorial http://blog.manula.org/2011/05/writing-simple-web-server-in-c.html**

http://www.ibm.com/developerworks/systems/library/es-nweb/

Open a TCP socket on port 80, start listening for new connections, implement [this](http://www.w3.org/Protocols/rfc2616/rfc2616.html). Depending on your purposes, you can ignore almost everything. At the easiest, you can send the same response for every request, which just involves writing text to the socket.

1. Open port 80 for listening
2. When contact is made, gather a little information (get mainly - you can ignore the rest for now)
3. Translate the request into a file request
4. Open the file and spit it back at the client
5. Get your basic TCP sockets layer running (listen on port/ports, accept client connections and send/receive data).
6. Implement a buffered reader so that you can read requests one line (delimited by CRLF) at a time.
7. Read the very first line. Parse out the method, the request version and the path.
8. Implement header parsing for the "Header: value" syntax. Don't forget unfolding folded headers.
9. Check the request method, content type and content size to determine how/if the body will be read.
10. Implement decoding of content based on content type.
11. If you're going to support HTTP 1.1, implement things like "100 Continue", keep-alive, chunked transfer.
12. Add robustness/security measures like detecting incomplete requests, limiting max number of clients etc.
13. Shrink wrap your code and open-source it :)