Introduction

You’re a programmer at a network startup and your goal is to write a web proxy. A web proxy is a program that, as discussed in class, reads a request from a browser, forwards that request to a web server, reads the reply from the web server, and forwards the reply back to the browser. You'll be writing a web proxy to learn about how to structure servers. For this project you'll start simple; in particular your proxy need only handle a single connection at a time. It should accept a new connection from a browser, completely handle the request and response for that browser, and then start work on the next connection. (A real web proxy would be able to handle many connections concurrently.)

Design Requirements

Your proxy will speak a subset of the HTTP/1.1 protocol, which is defined in RFC 2616. You're only responsible for a small subset of HTTP/1.1. Your proxy should satisfy these requirements:

- GET requests work and images and arbitrary length binary files are transferred correctly.
- Your proxy should properly handle Full-Requests (RFC 1945 section 4.1) up to 65535 bytes. You may close the connection if a Full-Request is larger than that.
- You must support URLs with a numerical IP address instead of the server name (e.g. http://128.181.0.31/).
- You are not allowed to use `fork()` or threads/multiprocesses of any kind.
- You may not allocate more than 30MB of memory.
- You cannot have more than 32 open file descriptors.
- Your proxy should correctly service each request if possible. If errors occur it should close the connection and then proceed to the next request. If an error occurs from which the proxy cannot reasonably recover, the proxy should print an error message on the standard output and call `exit(1)`. There are not many non-recoverable errors; perhaps the only ones are failure of the initial `socket()`, `bind()`, `listen()` calls, or a call to `accept()`. Your proxy should never dump core except in situations beyond your control (e.g. hardware/OS failure) and your proxy must not crash for any kind of signal errors other than hardware failures.
- You may NOT use any predefined libraries (e.g. `libwww`, `libhttpX`) or parsers or copy others’ code or code from the web. If you are in doubt *ask*. Using libraries that have not been cleared with the instructor will result in *significant grade deductions*.
- Your proxy (both stages) should be written in at most 300 lines of C (excluding comments) and it should compile and run on AFS.
- **The standard University Academic Integrity Policy applies. Violations will result in a grade of zero for the project.**

You do **not** have to worry about correct implementation of any of the following features; just ignore them as best you can:

- POST or HEAD requests.
- URLs of any type other than http.
- HTTP-headers (RFC section 4.2).

Running and testing your proxy

Your proxy program should take exactly one argument, a port number on which to listen. For example, to run the proxy on port 2000:

```
$ gcc –Wall –o wproxy wproxy.c        ## this is how we’ll compile (on AFS)
$ ./wproxy 2000 >out.txt              ## output goes to out.txt
Ctl-C                                 ## how we’ll stop it
```

As a first test of the proxy you should attempt to use it to browse the web. Set up your web browser to use one of the class machines running your proxy as a proxy and experiment with a variety of different pages.

Submit / Deliverables

Your proxy will be built in two stages. In stage one you will write a proxy (all in one C file called `proxy.c`) which we will compile using `gcc -Wall` (like the example TCP servers). In stage two you will be provided with a skeleton proxy (just like in a real world software company that is manufacturing high speed low latency routers e.g. Cisco) where you will “merge” your stage 1 code into the skeleton and make it work. Your final submission will be 3 files: (1) your stage 1 `proxy.c`, (2) your stage 2 skeleton proxy, `wproxy.c` and (3) a PDF report (2 pages or less) that outlines your design and documents your assumptions.

Grading / Testing

We will test your proxy using one or more of the following tests:

Ordinary fetch	Fetch a normal web page.
Split request	Request in more than one ‘chunk’ (connection)
Large request/resp	Requests of size 65535 bytes or response of size about 200MB
Bad Connect	URL to a bad port or bad host (e.g. `blahblah.com:2222`)
Malformed request	HTTP request that is syntactically incorrect
Huge request	Requests larger than 65535 bytes (of up to 1MB)
Stress test	We will test your proxy with several high-speed requests

We may use some combination of these tests, including some unspecified tests that are within HTTP protocol. (Think of this as a demo to your ‘startup’ venture capitalist for next round funding. Just as at a startup, you will have to design your own tests to mirror how we will test your code.) Your final grade will depend on your design and the performance of your code. Roughly speaking, proxies that pass our tests and remain standing will earn full marks, each failed test gets a 5-10% deduction, proxies that don’t even do Ordinary Fetch will get zero.

FAQ

I will post a FAQ for clarifications/questions; this will be the final arbiter of requirements/grading.