Eiffel HTTP Server

PROJECT PLAN

Semester project

Project period:	Winter Semester 2011
Student name:	Florian Besser
Status:	6th Semester Informatik Bsc
Email address:	fbesser@ethz.ch
Supervisor name:	Scott West

1. PROJECT DESCRIPTION

Overview

The HTTP specification is one which is utilized by all web-browsers and servers alike. More than this, it is a well-known and reviewed specification. This makes it a fantastic entry-point for the construction of a real-world application (web-server). The proposed server would be implemented in Eiffel, conform to an agreed upon portion of the HTTP specification (RFC 2616), and use concurrency to handle multiple connections and facilitate increased throughput.

Scope of the work

Prerequisites for developing the Server:
EiffelStudio, Version 6.6, together with several Libraries which are automatically supplied together with EiffelStudio (Example: EiffelNet).

Prerequisites for running the Server:

- Any operating system that supports EiffelStudio and it's libraries, so the sources can be compiled into an executable format.
- The system must provide a workspace for the server, namely it must give the server read-permission to it's own configuration files.
- The system must also allow the server to listen on ports, to open new ports and to send messages through those ports.
- Furthermore the system must allow the server to create threads or (in case SCOOP is used) SCOOP-based threads.

The idea is to create a HTTP server which can later be extended to work as a webserver. In order to provide a solid base for such an extension, the most important types of requests must be answered correctly. For a specification of these requests, see RFC 2616, Sect. 2

Intended results

Base Functionality (HTTP)
The HTTP Server complies with a good part of the the rules of HTTP 1.1, which means the Server can handle OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE and CONECT requests, and the answers to these requests are understood and correctly handled by major browsers like Firefox, Opera, Chrome, Safari, IE etc.
Additional (but not required) results would be that the Headers sent with the requests and answers are also handled according to HTTP 1.1.

Those types of requests the server complies with are supposed to be error-free in general cases and have a low error rate in special cases.

Base Functionality (Server Code / Execution)
The Server is multithreaded, preferably uses SCOOP.
The Server provides a good speed, meaning it is supposed to access more than 100 medium-sized static HTML pages per second. In a real-world environment the Server is supposed to reach 5% of the throughput-rate of an Apache webserver.
The server is supposed to run stable and smoothly, serving a constant rate of pages per second.

Base Functionality (Extensions / Entities)
MIME-Support (required to answer many requests, such as GET)
The configuration options of the server allow a flexible deployment and are easily extendable.

Configuration files for:
- Logging
- Virtual hosts (more than one domain on a server)
- Error reporting
- Directory options
- Handling of persistent connections (keep-alive etc.)
- Resource control (Number of spawned worker threads, and behavior of the Server under high load)
- MIME types

2. BACKGROUND MATERIAL

Reading list

Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616)
Found at: http://www.w3.org/Protocols/rfc2616/rfc2616.html
Nienaltowski P.: Practical framework for contract-based concurrent object-oriented programming
Found at: http://se.ethz.ch/people/nienaltowski/papers/thesis.pdf

3. PROJECT MANAGEMENT

Objectives and priorities

HTTP 1.1 Compliance	Priority: Critical
Avoidance of Starvation / Dead Lock (Threads)	Priority: High
Speed	Priority: Medium
Extendable configuration	Priority: Medium

Criteria for success

- Percentage of HTTP protocol implemented
- Percentage of Threads efficiently used (efficient means: given work without undue delay, Thread does not starve, and does not lock with other Threads)
- Delivered static pages per second
- Configuration possibilities in comparison to Apache

Method of work

Bi-weekly meetings with supervisor, and contact by e-mail to quickly address any problems that arise during the course of the project.

Quality management

Documentation

Minimal documentation of configuration files (since configuration files are equal to those of the Apache webserver).

Documentation of the HTTP Server includes:
- Overview over main features
- Guide for easy setup under Windows and Linux (compile from source).
- Guide to configure a secure Server, allowing only minimal access without authentication.
- Guide to expand the server to handle PHP and other extensions.
- Guide to handle DDoS attacks or operate under heavy load.

Documentation of the Codebase includes:
- Important design decisions
- Flowcharts on program behavior
- (Optional) UML Charts on code execution

Test Suites

We use three test suites:

- Hand-crafted: A suite that is supposed to cover all the base cases of the implemented features. The Server responses can be compared with the HTTP 1.1 definition and the responses of the Apache webserver.
- Wildlife: I operate a webserver with about 35'000 hits per day, and many more requests. These requests will be thrown at our Server to test for crashes, memory leaks and instability. However, since the amount of data is quite huge, analysis of every single request and answer will not likely be possible.
- Random: We can randomly generate requests with headers, and then compare the result returned for our Server with the HTTP 1.1 definition. The answer should not be compared to the Apache webserver, since Apache is fault-tolerant, and will often change the request in order to allow faulty clients to connect.
  Example:
  GET * HTTP/1.1 will return the same as GET / HTTP/1.1, but this is incorrect.

4. PLAN WITH MILESTONES

Project steps

1. Read and understand HTTP, Installation of necessary tools as well as an Apache webserver as a loose reference webserver.
2. Implement a single-threaded HTTP Server, that handles most of the protocol correctly as defined in "intended Results"
3. Make the Server multithreaded where possible, add various configuration options, such as on which port the server listens, or which folder is the "document root"
4. (Optional) Try converting the multi-threaded model into a model using SCOOP.
5. Various speed improvements and other performance-based accelerations.
6. Writing report, final clean-up of documentation, test suite etc.

Deadline

22. May 2011, around 2 weeks before end of spring semester 2011

Tentative schedule

1. 14. February 2011
2. 15. March 2011
3. 11. April 2011
4. 25. April 2011
5. 8. May 2011
6. 22. May 2011

REFERENCES

[1] Chair of Software Engineering: *Semester-/Diplomarbeiten*; Online at: http://se.inf.ethz.ch/projects/index.html, consulted in October 2002.

[2] Bertrand Meyer: *Object-Oriented Software Construction, 2nd edition*, Prentice Hall, 1997.