IMPLEMENTAION OF LIGHT-WEIGHTED WEB SERVER

MAIN PROJECT REPORT

submitted by

ABHILASH A K

ARUNRAJ M R

HARIKRISHNAN R

SHIJITH T

SHRUTHI V

for the award of the degree

of

Bachelor of Technology

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GOVERNMENT ENGINEERING COLLEGE SREEKRISHNAPURAM

PALAKKAD

July 2010

CERTIFICATE

This is to certify that the main project intermediate report entitled **IMPLEMENTATION OF LIGHT-WEIGHTED WEB SERVER** submitted by **ABHILASH A K**, **ARUNRAJ M R**, **HARIKRISHNAN R** and **SHIJITH T**, **SHRUTHI V**to the **Department Of Computer Science and Engineering, Government Engineering College, Sreekrishnapuram, Palakkad**, in partial fulfillment of the requirement for the award of B-Tech Degree in Computer Science and Engineering is a bonafide record of the work carried out by them.

Project Co-ordinator Internal Guide Head of the Department

Place: Sreekrishnapuram

Date:

Acknowledgment

First and foremost we wish to express our wholehearted indebtedness to God Almighty for his gracious constant care and blessings showered over us for the successful completion of the project.

We wish to express our profound gratitude to our guide, **Sijo S Vadakkan** for guiding and rendering help during the completion of the project.

We would like to express our deep gratitude to our internal guide **Dr. K Najeeb**, Computer Science and Engineering Department, and **Dr. P C Reghu Raj**, Head of Department of Computer Science and Engineering, for guiding and helping us throughout the project. We express our sincere thanks to all the other teaching and non-teaching staff of the department for their valued guidance. We are also grateful to our friends who helped us with their timely suggestions and supports during this endeavor.

Last but not the least, the thanks for the support and encouragement provided by our parents is inexpressive through words.

Table of Contents

List of Figures

List of Abbreviations

HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
URI	Uniform Resource Identifier
URL	Uniform Resource Location
URN	Uniform Resource Name
UA	User Agent
O	Original server
TCP/IP	Transmission Control Protocol/Internetworking Protocol

Abstract

The number of people using internet is growing day by day. So is the load on the web server processing these requests. This demands the creation of a web server with high load balancing capacity. The project aims at implementing a web server with following functionalities:

- **Standard HTTP web server**

  Implementing requierments specified in RFC1945 (HTTP/1.0)

  - Establish connection.
  - Handle URL.
  - Process HTTP requests and messages.

- **Efficient use of system resources**

  Create cheep processes to handle incoming connections with fast context switches. Effective use of physical memory with run queues, scheduling and garbage collection.

- **Handle high load**

These are implemented in Erlang.

CHAPTER 1

Introduction

A web server is a computer program that delivers (serves) content, such as web pages, using the Hypertext Transfer Protocol (HTTP), over the World Wide Web. The term web server can also refer to the computer or virtual machine running the program.

The primary function of a web server is to deliver web pages to clients. This means delivery of HTML documents and any additional content that may be included by a document, such as images. A client, commonly a web browser or web crawler, initiates communication by making a request for a specific resource using HTTP and the server responds with the content of that resource, or an error message if unable to do so. The resource is typically a real file on the server's secondary memory, but this is not necessarily the case and depends on how the web server is implemented.

Erlang is a general-purpose concurrent programming language and runtime system. The sequential subset of Erlang is a functional language, with strict evaluation, single assignment, and dynamic typing.It was designed by Ericsson to support distributed, fault-tolerant, soft-real-time, non-stop applications. The first version was developed by Joe Armstrong in 1986.It supports hot swapping thus code can be changed without stopping a system. processes communicate using message passing instead of shared variables, which removes the need for locks.

* **Why erlang ?**

Erlang is a lightweight concurrency model with massive process scalability independent of the underlying operating system is second to none. With its approach that avoids shared data, Erlang is the perfect fit for multicore processors, in effect solving many of the synchronization problems and bottlenecks that arise with many conventional programming languages. Its declarative nature makes Erlang programs short and compact, and its built-in features make it ideal for fault-tolerant, soft real-time systems. Erlang also comes with very strong integration capabilities, so Erlang systems can be seamlessly incorporated into larger systems. This means that gradually bringing Erlang into a system and displacing less-capable conventional languages is not at all unusual. This reflects in it's efficiency in handling parallel sessions above 4000, where traditional web servers in C and Java fails.

CHAPTER 2

Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It builds on the discipline of reference provided by the Uniform Resource Identifier (URI), as a location (URL) or name (URN), for indicating the resource to which a method is to be applied. HTTP allows basic hypermedia access to resources available from diverse applications.

HTTP is a request-response protocol standard for client-server computing. The client submits HTTP requests to the responding server by sending messages to it. The server, which stores content (or resources) such as HTML files and images, sends messages back to the client in response. These returned messages may contain the content requested by the client or may contain other kinds of response indications. A client is also referred to as a user agent (UA). Most HTTP communication is initiated by a user agent and consists of a request to be applied to a resource on some origin server. In the simplest case, this may be accomplished via a single connection (v) between the user agent (UA) and the origin server (O).

```
   request chain ------------------------>
UA -------------------v------------------- O
   <----------------------- response chain
```

Figure 2.1: HTTP communication

In between the client and server there may be several intermediaries. There are three common forms of intermediary: proxy, gateway, and tunnel. A proxy is a forwarding agent, receiving requests for a URI in its absolute form, rewriting all or part of the message, and forwarding the reformatted request toward the server identified by the URI. A gateway is a receiving agent, acting as a layer above some other server(s) and, if necessary, translating the requests to the underlying servers protocol. A tunnel acts as a relay point between two connections without changing the messages; tunnels are used when the communication needs to pass through an intermediary (such as a firewall) even when the intermediary cannot understand the contents of the messages.

```
   request chain ------------------------------------>
UA -----v----- A -----v----- B -----v----- C -----v----- O
   <----------------------------------- response chain
```

Figure 2.2: HTTP communication with intermediaries

The figure above shows three intermediaries (A, B, and C) between the user agent and origin server. A request or response message that travels the whole chain will pass through four separate connections. This distinction is important because some HTTP communication options may apply only to the connection with the nearest, non-tunnel neighbor, only to the end-points of the chain, or to all connections along the chain. Although the diagram is linear, each participant may be engaged in multiple, simultaneous communications. For example, B may be receiving requests from many clients other than A, and/or forwarding requests to servers other than C, at the same time that it is handling As request.

2.1 Request and Response

An HTTP client initiates a request. HTTP communication usually takes place over TCP/IP connections. The default port is TCP 80, but other ports can be used. This does not preclude HTTP from being implemented on top of any other protocol on the Internet, or on other networks. An HTTP server listening on that port waits for a client's request message. Upon receiving the request, the server sends back a status line, such as "HTTP/1.0 200 OK", and a message of its own, the body of which is perhaps the requested resource, an error message, or some other information.

2.1.1 Request message

The request message consists of the following:

- Request line, such as GET /images/logo.png HTTP/1.0, which requests a resource called /images/logo.png from server
- Headers allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers, with semantics equivalent to the parameters on a programming language method invocation.
- An empty line
- An optional message body

The request line and headers must all end with <CR><LF> (that is, a carriage return followed by a line feed). The empty line must consist of only <CR><LF> and no other whitespace.

2.1.2 Request methods

HTTP/1.0 defines three methods (sometimes referred to as "verbs") indicating the desired action to be performed on the identified resource. What this resource represents, whether pre-existing data or data that is generated dynamically, depends on the implementation of the server. Often, the resource corresponds to a file or the output of an executable residing on the server.

- HEAD

  Asks for the response identical to the one that would correspond to a GET request, but without the response body. This is useful for retrieving meta-information written in response headers, without having to transport the entire content.

- GET

  Requests a representation of the specified resource. Note that GET should not be used for operations that cause side-effects, such as using it for taking actions in web applications. One reason for this is that GET may be used arbitrarily by robots or crawlers, which should not need to consider the side effects that a request should cause. See safe methods below.

- POST

  Submits data to be processed (e.g., from an HTML form) to the identified resource. The data is included in the body of the request. This may result in the creation of a new resource or the updates of existing resources or both.

2.1.3 Response message

- Status-Line, consisting of the protocol version followed by a numeric status code and its associated textual phrase, with each element separated by SP characters. No CR or LF is allowed except in the final CRLF sequence.
- The response-header fields allow the server to pass additional information about the response which cannot be placed in the Status-Line. These header fields give information about the server and about further access to the resource identified by the Request-URI.
- CRLF
- message-body

2.2 Example session

Below is a sample conversation between an HTTP client and an HTTP server running on www.example.com, port 80.

- Client request:

  $POST/~hy556/cgi - bin/post - queryHTTP/1.0$

  $Accept : text/html, video/mpeg, image/gif, application/postscript$

  $User - Agent : Lynx/2.8.4libwww/5.4.0$

  $From : giannak@csd.uoc.gr$

  $Content - Type : application/x - www - form - urlencoded$

  $Content - Length : 150$

  $* * ablankline *$

  $org = Distributed\%20Systemsprofessor = Marazakisbrowsers = lynx$

  A client request (consisting in this case of the request line and only one header) is followed by a blank line, so that the request ends with a double newline, each in the form of a carriage return followed by a line feed.

- Server response:

  $HTTP/1.0200OK$

  $Date : Fri, 08Aug200308 : 12 : 31GMT$

  $Server : Apache/1.3.27(Unix)$

  $MIME - version : 1.0$

  $Last - Modified : Fri, 01Aug200312 : 45 : 26GMT$

  $Content - Type : text/html$

  $Content - Length : 2345$

  $* * ablankline *$

  $< HTML > ...$

  A server response is followed by a blank line and text of the requested page.

CHAPTER 3

Webserver Design

The web server runs as a system daemon.The web server is typical of a large number of programs. It involves the interaction between Erlang an some entity operating in the outside world.As far as an Erlang process is concerned, all other objects in it's universe are Erlang processes. The only thing that an Erlang process knows how to do, is how to to send to and receive messages from other Erlang processes. A web server is therefore a simple process that receives a message containing a request to read a page and which responds by sending that page to the process which requested the page. In order to do this in a consistent manner we write a device driver which we use to interface the external world with Erlang.It comprises of three modules.

3.1 Three modules

The three modules are tcp driver, http driver and web server.

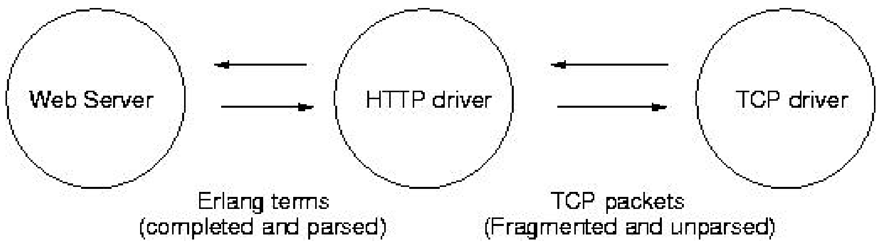


Figure 3.1: Web server model

- **TCP driver**

  Receives fragmented packets and pass the unparsed fragments to http driver. Send server responce packet to client.

- **HTTP driver**

  HTTP requests, are not simple Erlang terms, but are actually TCP streams and they are segmented(fragmented). For this reason we introduce an intermediary process (called a middle-man). The middle man is a process whose only job is to recombine fragmented TCP packets, parse the packets assuming they are HTTP requests, and send the requests to the web server. This also forms the response from erlang terms of web server.

- **Web server**

  Find the requested file and parse the page to create the response.

CHAPTER 4

Benchmarking

A web system consists of a web server, a number of clients, and a network that connects the clients to the server. The protocol used to communicate be- tween the client and server is HTTP . In order to measure server performance in such a system it is necessary to run a tool on the clients that gen- erates a specific HTTP workload. Currently, web server measurements are conducted using bench-marks which simulate a fixed number of clients.

4.1 TSung

Tsung (formerly IDX-Tsunami) is a distributed load testing tool. It is protocol-independent and can cur rently be used to stress HTTP, WebDAV, SOAP, PostgreSQL, MySQL, LDAP, and Jabber/XMPP servers. Tsungs main strength is its ability to simulate a huge number of simultaneous user from a single CPU. Tsung is developed in Erlang and this is where the power of Tsung resides. Erlang is a concurrency-oriented programming language. Tsung is based on the Erlang OTP (Open Transaction Platform) and inherits several characteristics from Erlang:

- Performance: Erlang has been made to support hundred thousands of lightweight processes in a single virtual machine.
- Scalability: Erlang runtime environment is naturally distributed, promoting the idea of processs location transparency.
- Fault-tolerance:Erlang has been built to develop robust, fault-tolerant systems. As such, wrong answer sent from the server to Tsung does not make the whole

running benchmark crash.

4.1.1 HTTP related features

- HTTP/1.0 and HTTP/1.1 support
- GET, POST, PUT, DELETE and HEAD requests
- WWW-authentication Basic
- User Agent support
- Any HTTP Headers can be added
- Proxy mode to record sessions using a Web browser

4.1.2 Advantages

Tsung has several advantages over other injection tools:

- High performance and distributed benchmark: You can use Tsung to simulate tens of thousands of virtual users.
- Ease of use: The hard work is already done for all supported protocol. No need to write complex scripts. Dynamic scenarios only requires small trivial piece of code.
- Multi-protocol support: Tsung is for example one of the only tool to benchmark SOAP applications.

4.2 Running

A typical way of using tsung is to run: tsung -f myconfigfile.xml start. The command will print the current log directory created for the test, and wait until the test is over. A typical erlang.xml configuration file we used is:

```
<?xml version=”1.0”?><tsung loglevel=”notice” version=”1.0”>

<clients>
<client host=”localhost” use_controller_vm=”true”/>
</clients>

<servers>
<server host=”localhost” port=”4501” type=”tcp”/>
</servers>

<monitoring>
<monitor host=”localhost” type=”snmp”/>
</monitoring>

<load>
<arrivalphase phase=”1” duration=”15” unit=”minute”>
<users interarrival=”1” unit=”second”/>
</arrivalphase>
</load>

<options>
<option type=”ts_http” name=”user_agent”>
<user_agent probability=”80”>Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.8)
Gecko/20050513
```

```
Galeon/1.3.21</user_agent>
<user_agent probability=”20”>Mozilla/5.0 (Windows; U; Windows NT 5.2; fr-FR;
rv:1.7.8) Gecko/20050511 Firefox/1.0.4</user_agent>
</option>
</options>
<sessions>
<session name=”http-example” probability=”100” type=”ts_http”>
<request> <http url=”/index.html” method=”GET” version=”1.1”/> </request>
</session>
</sessions>
</tsung>
```

Calculate the normal duration of the scenario and use the interarrival time between users and the duration of the phase to estimate the number of simultaneous users for each given phase. Launch benchmark with your first application parameters set-up: tsung start. Wait for the end of the test or stop by hand with tsung stop. The statistics are updated every 10 seconds. For a brief summary of the current activity, use tsung status. Analyze results, change parameters and relaunch another benchmark.

4.3 Statistics and reports

Available stats

- request Response time for each request.
- page Response time for each set of requests (a page is a group of request not separated by a think- time).
- connect Duration of the connection establishment.
- reconnect number of reconnection.
- size_rcv Size of responses in bytes.
- size_sent Size of requests in bytes.
- session Duration of a users session.
- users Number of simultaneous users.
- connected Number of simultaneous connected users.

4.4 Tsung Plotter

Used to compare different tests runned by Tsung. tsplot is able to plot data from several tsung.log files onto the same charts, for further comparisons and analyzes. You can easily customize the plots you want to generate by editing simple configuration files.
Example of use:
tsplot "First test" firsttest/tsung.log "Second test" secondtest/tsung.log -d outputdir

4.4.1 Comparison with Apache2

TSung test results for apache2 and erlang are ploted using tsung plotter. The various graphs are shown below. X-axis shows time. Y-axis shows various factors.

HTTP requests per second

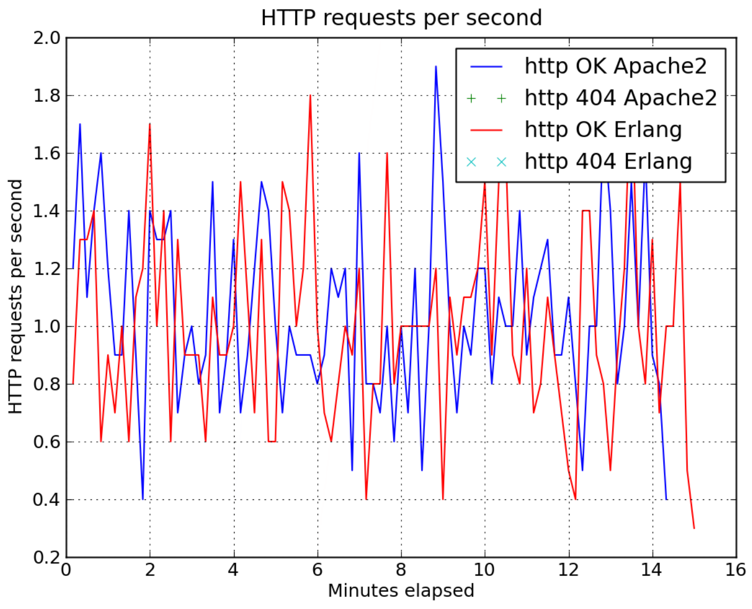


Figure 4.1: HTTP requests

Mean duration of request

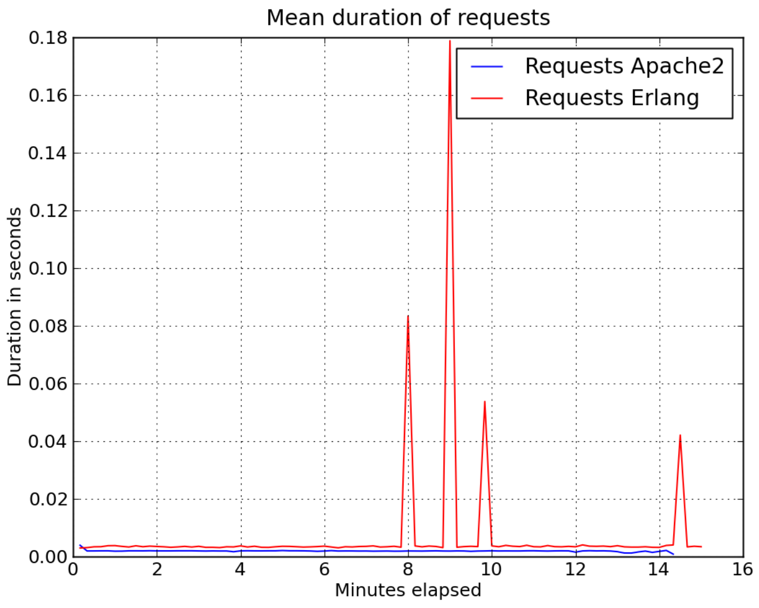


Figure 4.2: Mean request duration

Ending users per second

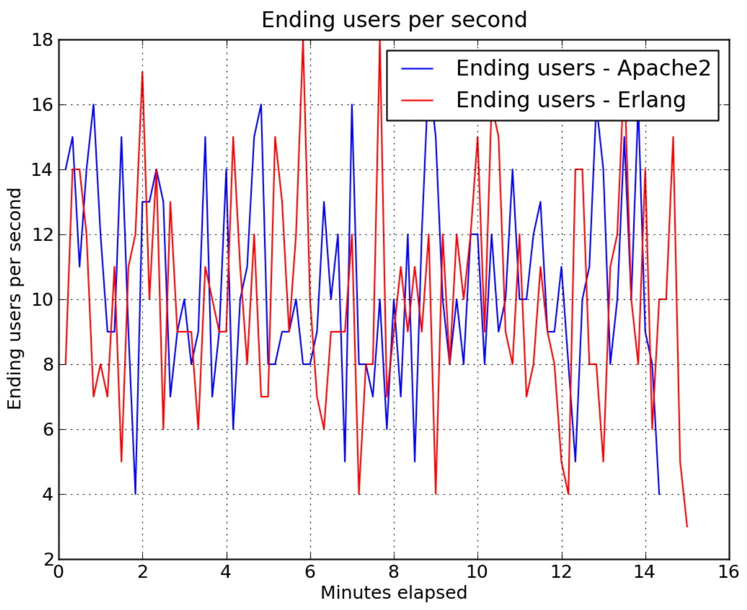


Figure 4.3: Finished users

CHAPTER 5

Experimental Validation and Result

5.1 Environment

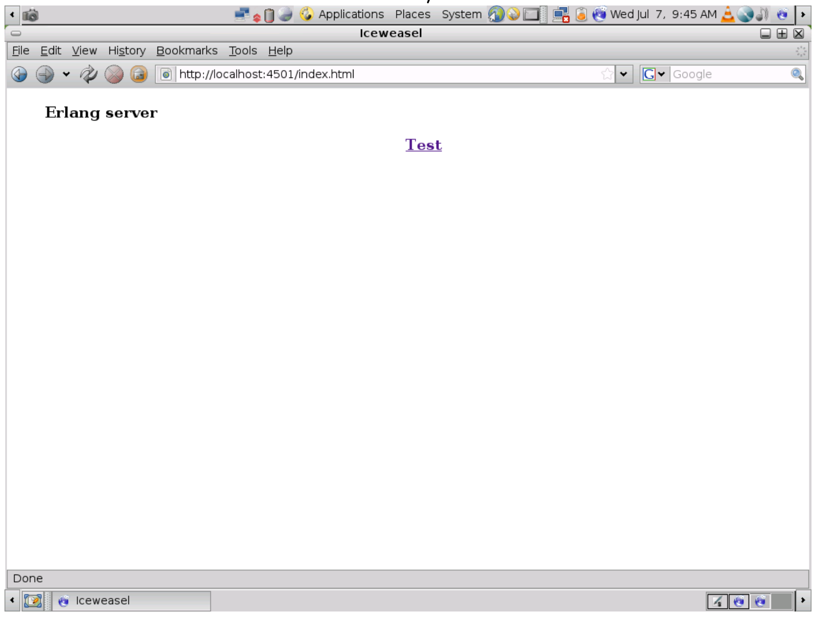
The development environment is Linux, which is open source and free. The system requires the following to be installed : Erlang 5.6, web-browser.
The web server is started using the command:
./webserver.sh start
Starting Erlang with the flag -heart starts Erlang in heartbeat mode. In heartbeat mode an external program monitors the Erlang system. If the Erlang system dies the system is restarted by evaluating the command in the environment variable ERLANG_HEART. The value of the environment variable is $PA/web_server.sh start and so the program just gets restarted.
Starting Erlang with the flag -detached starts Erlang in detached mode. In detached mode Erlang runs silently in the background without a controlling terminal.
To stop the web server run
./webserver.sh stop

5.2 Input and Output

Input - URI is entered in the browser address bar.
Output - Page is displayed if exists and an error message otherwise.

Screenshots of browser window showing output are shown below:

localhost:4501/index.html

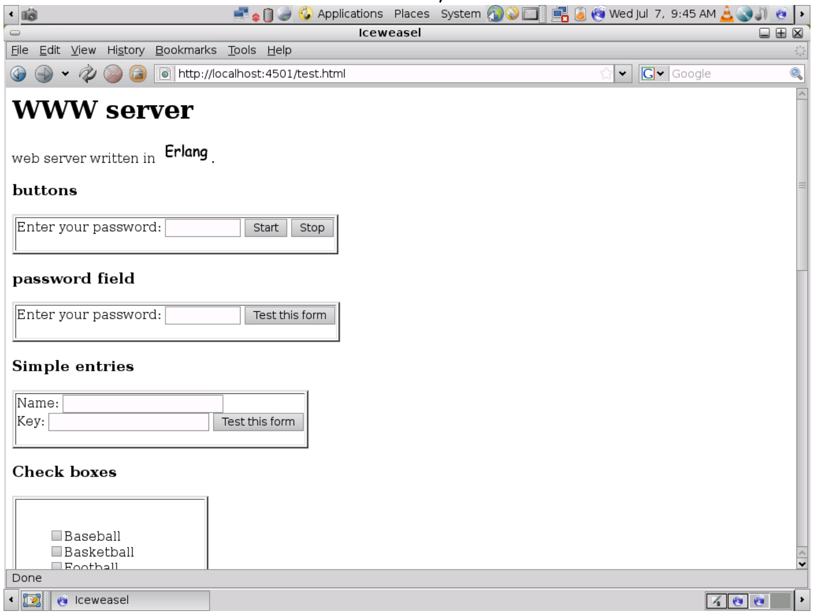

Applications Places System
Wed Jul 7, 9:45 AM
Iceweasel
File Edit View History Bookmarks Tools Help
http://localhost:4501/index.html
Google
Done
Iceweasel

Erlang server

Test

Figure 5.1: index.html

localhost:4501/test.html


Applications Places System
Wed Jul 7, 9:45 AM
Iceweasel
File Edit View History Bookmarks Tools Help
http://localhost:4501/test.html
Google
Done
Iceweasel

WWW server

web server written in Erlang.

buttons

Enter your password: [ ] Start Stop

password field

Enter your password: [ ] Test this form

Simple entries

Name: [ ]
Key: [ ] Test this form

Check boxes

☐ Baseball
☐ Basketball
☐ Football

Figure 5.2: test.html

CHAPTER 6

Conclusion and Future work

6.1 Conclusion

HTTP/1.0 web server implemented in erlang is working successfully as expected.The benchmark indicates the perfomance reaches levels of server like apache2. With it's inherent features erlang makes the web server very much concurrent and gives it the ability to handle heavy loads.

6.2 Future Work

- HTTP/1.1
  The current web server can be extended to support HTTP/1.1 by adding new function.
- PHP engine
  PHP parser modules need to be build and integrated to the web server for the processing of web pages on demand.
- Database accessing
  It demands the creation of modules for accessing database and retrieve data form it .
- HTTPS
  Cryptographic functions should be added for method https.

Bibliography

[**1**] Francesco Cesarini, Simon Thompson, *Erlang Programming*,First Edition, June 2009.

[**2**] Joe Armstrong, *Pragmatic Programming Erlang : Software for a Concurrent World, rdJuly 2007.*

[**3**] *http://tsung.erlang-projects.org*, Visited last on : July 2010

[**4**] Joe Armstrong, *Concurrency Oriented Programming in Erlang.*, 17 February 2003.

[**5**] John Yannakopoulos, *HyperText Transfer Protocol: A Short Course.*, August 2003.

[**6**] IETF, *RFC1945*