Assignment 1

PDF version of this page - for printing. Due to inconsistencies in browsers displaying the contents of this page (especially newline characters in sample commands), please print using the PDF version of this page.

Part I: Apache web server configuration and administration (30%)

Your task is to set-up and maintain a web server to support the teaching within Pie-in-the-Sky University's School of IT. This School runs 15 units in each of the 2 semesters every year. The average enrolment for each of the units is approximately 70 students. The typical web usage for each of these units is similar to that of B336 at Murdoch University.

You must set-up at least the following **basic functionalities**:

i. The server must work (obviously).
ii. There must be a restricted-access directory called "staff" on the top level of the server's web-site. Access to this directory and any of its sub- directories is only allow to staff who supply valid usernames and passwords.
iii. The server must log all requests from clients, enough to identify the address the requests came from, what browser made the request (if possible), when the requests were made, what resources were requested (if any), and status of the response.
iv. The web site is able to include simple dynamic server-side content.
v. A description of a basic set of procedures to monitor the server's performance so that appropriate adjustments can be made to re-configure it if necessary.

Beyond the basic functionalities, you should also add two **extra functionalities** that adds the web server's value to the School. Exactly what extra functionalities is up to your understanding of and expertise with your web server. The functionalities will be assessed on the value they add to the School, and the technical understanding required for the new configurations.

Part II: Implement a Web Client (30%)

Complete the following:

1. Implement a perl web client named `webclient.pl` that supports basic HTTP communication with a web server. Your client should accept a URL as a parameter. For example:

   ```
   webclient.pl http://www.murdoch.edu.au
   ```

   The client should retrieve the resource specified by the URL using the "GET" method. The client should display the whole response (status line + headers + content) on standard output, regardless of whether the retrieval succeeded or not.

   The behaviour of your web client when no parameter is given is unspecified. You may implement anything you believe appropriate. This behaviour will not be assessed.

2. Add to your web client in Part II (1) above by include support for command line options. Your client should accept an optional first parameter before the URL such that:

   i. If the first parameter is "-s", for example:

      ```
      webclient.pl -s http://www.murdoch.edu.au
      ```

      then the client should retrieve the resource specified by the URL using the "HEAD" method. The client should display the status line (note: display the status line **ONLY**) on standard output. The status line should be in standard form:

      <HTTP-version> <code> <code-string>

```
<HTTP version> <code> <code string>
```

ii. If the first parameter is "-c", for example:

```
webclient.pl -c http://www.murdoch.edu.au
```

then the client should retrieve the resource specified by the URL using the "GET" method. If the retrieval is successful, display the contents (note: display the contents **ONLY**) on standard output. If it fails, display the status code string (note: display the status code string, or also called status message, **ONLY**) on standard output.

iii. If the first parameter is "-h", for example:

```
webclient.pl -h http://www.murdoch.edu.au
```

then the client should retrieve the resource specified by the URL using the "HEAD" method. If the retrieval is successful, display the headers (note: display the headers **ONLY**) on standard output. If it fails, display the status code string (note: display the status code string, or also called status message, **ONLY**) on standard output.

iv. If the first parameter is "-f", for example:

```
webclient.pl -f result.html http://www.murdoch.edu.au
```

then the client should retrieve the resource specified by the URL using the "GET" method and save the contents (note: save the contents **ONLY**) in the specified file. If the file exists, overwrite the file. Save the content regardless of whether the retrieval is successful or not.

3. Add to your web client in Part II (1) above by including support for HTTP Basic Authentication. Your web client should behave the same as when a standard browser attempts to access a password protected site. It should prompt the user for a username and password. If the username and password is correct, then the final request will be successful.

   The following is a description of HTTP Basic Authentication:

   When making a request for a resource, if the client receives a response with status code "401", and the response contains a header:

   ```
   WWW-Authenticate: Basic realm="ABC"
   ```

   where "ABC" can be any string, then it should prompt the user for a `<username>` and a `<password>`. The prompt should be the same as the following example:

   ```
   Logging into ABC
   Username:xyz
   Password:012
   ```

   where "xyz" and "012" are typed in by the user. The script should then make another request to the same URL as normal except with an additional request header of the form:

   ```
   Authorization: Basic <base64-string>
   ```

   where `<base64-string>` is the Base-64 encoding of the string "`<username>:<password>`". For example, if the user supplied the username "xyz" and the password "012", then the Base-64 encoding of "xyz:012" is "eHl6OjAxMg==". Therefore the request should contain the header:

   ```
   Authorization: Basic eHl6OjAxMg==
   ```

   The script should display the result of this second request as stated in Part II (1).

   HTTP Basic Authentication is an official standard, so your web client must be able to authenticate with any standard web server using this authentication scheme. You may set-up your Apache web server in Part I to test your client, or you may test it by accessing the protect resource http://www.it.murdoch.edu.au/units/b336/restricted/whatisthesecret.txt (username "knock" and password "letmein"). Note, however, your final submission may not be tested with this particular resource, and this particular username-password combination.

Further details of HTTP Authentication is available from the official RFC 2617, but you are not required to implement anything beyond what is described above.

The behaviour of your web client when given the above options in combination is unspecified (eg. what happens if 2(iii) and 2(iv) happens in combination). You may implement anything you believe appropriate. This behaviour will not be assessed. Each requirement will be tested in isolation.

Your source code will be assessed. Read the "Source Code" section of this link to guidelines for programming for a description on what to consider when writing code.

You may use and modify the example Perl code given in lectures, labs and the unit reader, with appropriate acknowledgements in your submission. The code you borrowed from these sources will **not** be assessed, only the sections you add or modify.

Part III: Implement a Web Server (35%)

Complete the following:

1. Implement a web server in perl called `webserver.pl` which is able to handle HTTP requests from any standard web client, and return the requested resource (your server will be tested on serving plain text, HTML, and GIF images). Your server can accept one optional parameter corresponding to the port number to use. Your server should use port 8080 if no parameter is supplied. Your server should support the GET and HEAD methods defined by HTTP. You server must also respond with the error responses if it encounters the following situations:

    i. Request is not a valid HTTP request.
    ii. The URL is not found.
    iii. The URL has moved to a new location.

2. Add to your web server in Part III (1) by supporting the standard HTTP method called OPTIONS. When your server receives a request with the OPTIONS method, it should send back a response with header "Allow". The values of the "Allow" header should be comma-delimited strings corresponding to the HTTP methods supported by your server. For example:

    ```
    Allow: GET, HEAD, OPTIONS
    ```

    The status code of the response should be "200 OK". Your server should ignore the resource part of the request line in the request (it should **not** sent back an error status code if the resource is not available).

3. Add to your web server in Part III (1) by allowing configurations to be read from a configuration file. The configuration file name should be to the server as a parameter "-f". For example:

    ```
    webserver.pl -f config.txt
    ```

    If a configuration file is used, the server should not accept the port as a command line parameter. The configuration file should be in the form of lines of directives. Each line can only have one directive, and the lines have the form:

    ```
    <Directive>: <comma-separated values>
    ```

    The following directives should be supported in the configuration file:

    i. `Port`: the port number to use.
    ii. `AddType`: associate a mime-type string with a file extension.
    iii. `DocumentRoot`: the directory out of which to serve documents.
    iv. `DirectoryIndex`: index files (eg. index.html) to use when the url is a directory.

    Here is a link to a sample configuration file.

    The configuration file should be processed from top to bottom. Only the `AddType` directive may appear more than once. Any line with "#" as the first non-space character are comments and should be ignored. If the configuration file is not in the correct format, the server should stop with the error message:

    ```
    webserver.pl: Error processing <filename> line
    <line#>
    ```

Your source code will be assessed. Read the "Source Code" section of this link to

Your source code will be assessed. Read the "Source Code" section of this link to guidelines for programming for a description on what to consider when writing code.

You may use and modify the example Perl code given in lectures, labs and the unit reader, with appropriate acknowledgements in your submission. The code you borrowed from these sources will **not** be assessed, only the sections you add or modify.

Part IV: Learning to Learn (5%)

Read the *Learning to Learn* document, and complete the information in the two forms:

a. Plans. (2%)
b. Evaluation of plans and methods. (3%)

Marks Breakdown

Here is a link to a description of how the marks will be broken down.

Submission Requirements

Your submission should include the following:

a. B336 Assignment Cover Sheet. All submissions (Internal and External on all campuses) must include this cover sheet. External students must complete this assignment sheet as well as the cover sheet sent to you by the External Studies Office.
b. A statement of what you have completed - this is essential to guide the marking. Functionalities not stated as completed will not be assessed. Incomplete functionalities misleadingly stated as complete will incur heavy penalities.
c. Part I:
    - A working version of the Apache web server configured to do satisfy the stated functionalities, installed in and run from your home area on `gryphon.murdoch.edu.au`. Although having a full web site is not necessary in this assignment, you need to include enough example files (such as HTML files) in your directories to demonstrate that the requirements are satisfied.
      Do not to have a server daemon running at all times. You will be notified when you need to turn on your server to have it assessed.
    - Copies of your configuration files submitted on floppy disk.
    - A print-out of (only) the sections in the server configuration files that you have modified, with comments.
    - A description of procedures for basic functionality (v) - maximum 1 page.
    - Extra documentation to justify and describe extra functionalities.
d. Parts II & III:
    - Copies of your scripts submitted on floppy disk, AND residing in your home area on `gryphon.murdoch.edu.au`.
    - A print-out of the source code for your scripts.
e. Part IV:
    - A print-out of your plans and evaluation of plans.

Submission Dates

Submit a copy of your plans (part IV a) directly to your tutor **before or during week 4**. Internal students may do so during your lab sessions. External students may email a copy directly to your tutor. External students should ensure you receive and save the email message sent by your tutor acknowledging he/she has received your plans.

The whole assignment (including part IV a) is due at **12noon Tuesday 1st April 2003 (week 6)**. Internal students must submit their assignments to the School of IT's assignment boxes. External students must submit through the External Studies Office.

Learning Objectives:

As with all assignments, this assignment is to assess how well you have achieved the unit's learning objectives. This assignment is principally focused Learning Objectives 1 (understanding technologies), 2 (writing software) and 3 (constructing solutions). Part IV is to assess your progress in Learning Objective 4 (self-learning). To do well in this assignment, you will need to show good performance in these objectives.

Errata

This page is subject to change based on errors found. Any errors found and corrected will be posted to the "Corrections to Unit Materials" group in the unit's discussion

forum. If you print out a copy of this page, please follow the developments in that group for corrections.

Document author: H.L. Hiew, Unit Coordinator
Last Modified: Friday, 21-Mar-2003 12:57:22 WST