Arachno

Random/Raw unformatted ramblings/ideas

Regsister function calls on request, with a paremeter to get the request details(i.e. method, url, header, and body)

Option to register (pre-made)function that serves static files.

Option to register (pre-made)function that enables guessing filesnames from path(example ‘/user’ → ‘/user.html’ or ‘/contents’ → ‘/contents/index.html’)

Option to register (pre-made)function that tries to self resolve .htlmf files through the `<htmlf>` header.

Seperate event listener for general requests, and an event listener for parsing actual pages.

Functions the request listener gets the raw data about the request(method, url, header, and body) and decides what to do with that info. As for the parsing listener, arachno handles a lot by itself, allowing the programmer to do things like registering functions for specific pages or groups of pages, specify a value table for the htmlf formatting

Request/Response Life Cycle:

- Arachno recieves an HTTP request and parses it into a request struct(put link here).
- Request listener calls all registered(in the same order as registered) functions, passing a const pointer to the struct as a parameter. The return value of the function decides whether to continue to the next registered function, or treat the request as answered and stop.
- If the request has not been answered, parse the request further, look for a file in the same path as the URL path, Optionally try to see if a static HTML file of the same name(but with extention) exists and serve it, Optionally look for an index file, Optionally look for a .htmlf file. Check if the URL path is an entry in a separate table that map the path to an html/htmlf file and/or a code file and a list of listeners, if entry exists, call the appropriate listener(listeners for each method exist and also a general listener). If the entry is marked as htmlf, pre-parse the the htmlf file before handing control to the user.
- (Optional) If none of the functions have answered the request, show detailed 404 page.

.htmlf format

The htmlf format defines a dynamic html page which can be formatted in a `printf` like fashion.

Simple format

The “simple” format is exactly the same as printf, so for example a snippet that looks like that:

```
<div>
     <span id=”name”>%s<span>
     <span id=”age”>%d<span>
     <span id=”score”>%.3f<span>
</div>
```

Could be formatted in the exact same way as printf, by passing to a parsing function and supplying the values as variable arguments in the same order as they appear in the page.

Named format

The “named” format allows supplying a name at end of the format allowing for specifying values more easily as key-value pairs which also allows for a single value being able to be used in multiple times in the file without having . The format is %format%varname% for example:

```
<div>
     <span id=”name”>%s%username%<span>
     <span id=”age”>%d%age%<span>
     <span id=”score”>%.3f%finalscore%<span>
</div>
```

<htmlf> header

A .htmlf file starts with an `<htmlf>` header tag which allows specifying options for parsing the file. An empty header can be defined by using `<htmlf />`.

- <Format> Specifies the type of the format, options: Simple, Named. Default: Named.
- <DataSource> Specifies options for getting values for formatting from JSON resource. In htmlf simple format, an array is expected and the values formatted in the same order as they appear in the file(nth format specifier in the file is replaced with the nth value in the array). In htmlf named format, an object is expected and the values are
    - <Method> Specifies the HTTP method of the request. Default: GET.
    - <URL> Specifies the url to send a request to for the JSON resource.
    - <Object> Specifies the object in the JSON which contains the values. The default is the JSON object itself. For example, if the returned JSON file contains additional info like request status or additional data, and the values themselves are at object `result.data.user` than the value of <Object> would be `result.data.user`.
- <Version> The version of the htmlf format. Helps for backwards compatibility. Default: Latest stable htmlf version supported by the currently running version of arachno.

Formats

The supported % formats include everything printf supports, including all the specifiers(Except width and left-justified(-) which mean nothing in html) in addition to:

- %q – A string value from the URL's query string, supports all specifiers that %s supports. (Only supported in Named format). Unlike other formats, %q is supported in the <htmlf> header!
- %m – Same as %q but the values are taken from the body when the values are sent by a form as url encoded form data.
- %cardf – Inserts a .cardf template card. The value is a struct defined in [link to cardf struct] and can have it's own value list.
- %path, %upath

Future ideas:

loops

I wanna somehow allow loops. I can imagine having something like %for but like, what and how to go over???

Practical example:

```
<div id="store" class="cards">
%for%products%
<div id="prod%d%id%" class="card">
     <span class="prod_name">%s%name%</span>
     <hr />
     <img class="prod_img" src="%s%img_url%" />
     <p class="prod_summary">%s%summary%</p>
     <span class="prod_price">%.2f%price%</span>
</div>
%endfor%products%
</div>
```

Dynamic Compiling or Linking

Currently the plan is to have arachno as a library, and the user initializes the system and writes the main function and just links to the library, each code file for an htmlf page needs to be manually registered which also means a header file for each. Imagine if the website has 100 pages, that's 100 headers that defines basically the same thing, predefined functions for registering to the event listeners. And that also means somewhere in the code there's a huge function that has 100 lines of code that look something like:

```
...
register(home_page_GET);
register(products_page_GET);
register(products_page_POST);
register(register_page_GET);
register(register_page_POST);
...
```

which is kinda moronic. I would like to think of a way to maybe be able to define in the <htmlf> header something like <SourceFile>login_page.c</SourceFile> and the code would parse all the htmlf files to find all the compiled .o files of the source files that need to be linked to the project and somehow check if each has something like a onGET, onPOST, onDELETE, onRequest functions, and if yes register them, but I'm not sure how I actually want to do it so I don't really know how I would link them.
Or perhaps instead of that whenever a request for /login comes, the program goes into login.htmlf, sees <SourceFile>login_page.c</SourceFile> and only then links to it, jumps to the appropriate function (with a predefined name maybe?) that would handle the request?

No clue. Dad said to check out dlopen and dlsim, apparently can dynamically load an object file or something like that

Help it's 3AM.