PSet6: Web Server

1. Objetivos

- Familiarizarte con HTTP
- Aplicar técnicas familiares en contextos nuevos.
- Hacer la transición de C hacia programación web.

2. Honestidad Académica

Puede leer el código de conducta en: http://soporte.code-fu.net.ni/codigo-de-conducta-con-el-curso/

3. Evaluación

- Su trabajo en este PSet se evaluará a lo largo de cuatro ejes principalmente.

3.1. Alcance

- ¿En qué medida su código implementa las características requeridas por nuestra especificación?

3.2. Exactitud

- ¿Hasta qué punto su código está libre de errores?

3.3. Diseño

- ¿En qué medida está su código bien escrito (claramente, eficientemente, elegantemente, y/o lógicamente)?

3.4. Estilo

- ¿En qué medida es su código legible (comentado, sangrado, con variables adecuadamente nombradas)?

4. Preparándonos

Primero, acompañá a David en una vuelta por HTTP, el "protocolo"por el cual se comunican los buscadores y servidores web. ¡Mirá el video aquí!
Ahora, deberías revisar algunos de estos ejemplos de la semana 7, a través de los cuales introdujimos HTML, el lenguaje en el cual se las páginas web están escritas. ¡Mirá el video aquí!
Ademas, revisá algunos de estos ejemplos, con los cuales introdujimos CSS, el lenguaje con el cual las páginas web son estilizadas. ¡Mirá el video aquí!
Ahora, considera revisar algunos de estos ejemplos. Con estos ejemplos introdujimos los formularios HTML, los cuales ocupamos para enviar consultas GET a Google. ¡Mirá el video aquí!
Para tener otra perspectiva, acompañá también a Daven por otra vuelta por HTML. No podés perderte los bloopers al final. ¡Mirá el video aquí!
Finalmente, acompañate a Joseph y a Rob a echar un vistazo más cercano a CSS. ¡Mirá el video aquí!

5. Empezando

Vaya a su cuenta en cs50.io y ejecute el comando

```
update50
```

en la terminal para asegurarte workspace está actualizado.
Como en el Problem Set 5, este Problem Set viene con cierto código de distribución que necesitarás descargar antes de empezar. Ahora, ejecutá

```
cd ~/workspace
```

para navegar en tu directorio `~/workspace`. Ahora, ejecutá

```
wget http://cdn.cs50.net/2015/fall/psets/6/pset6/pset6.zip
```

para descargar un ZIP del distro de este Problem Set. Si luego ejecutás

```
ls
```

deberías ver que ahora tenés un archivo llamado `pset6.zip` en tu `~/workspace`. Descomprimilo ejecutando lo siguiente:

```
unzip pset6.zip
```

Si de nuevo ejecutás

```
ls
```

deberías ver ahora tenés también un directorio `pset6`. A continuación te invitamos a borrar el archivo ZIP con el siguiente comando:

```
rm -f pset6.zip
```

Ahora, entrá en ese directorio `pset6` al ejecutar lo siguiente.

```
cd pset6
```

Ahora, ejecutá

```
tree
```

(el cual es una variante jerárquica y recursiva de `ls`), y deberías ver que ese directorio contiene lo siguiente

```
.
|--- Makefile
|--- public
|     |--- cat.html
|     |--- cat.jpg
|     |--- favicon.ico
|     |--- helllo.html
|     |--- hello.php
|     |--- test
|          |--- index.html
|--- server.c
```

¡Rayos!, aún C. Pero, ¡algunas otras cosas también!
Echá un vistazo en `cat.html`. Simple, ¿no? Parece que tiene una etiqueta `img`, el valor de cuyo atributo `src` es `cat.jpg`.

A continuación, echemos un vistazo en `hello.html`. Notá cómo tiene un `form` que está configurado para enviar vía GET un campo `text` llamado `name` hacia `hello.php`. Tiene sentido, ¿verdad? Si no, tratá de dar otro vistazo al tutorial de `search-0.html` de la semana 7.

Ahora, miremos `hello.php`. Notá cómo es en su mayor parte HTML, pero dentro de su `body` hay un poco de código PHP:

```
<?= htmlspecialchars($_GET["name"]) ?>
```

La notación `<?=` significa "haga eco del siguiente valor aquí". `htmlspecialchars`, mientras tanto, es solo una función con un nombre horrible cuyo propósito en la vida es asegurar que caracteres especiales (incluso peligrosos) como `<` sean propiamente "evacuados" como "entidades" de HTML.
Visitá http://php.net/manual/en/function.htmlspecialchars.php para más detalles si tenés curiosidad.
De cualquier forma, `$_GET` es una variable "superglobal" dentro de la cual hay cualquier tipo de parámetros HTTP que fueron pasados vía GET hacia `hello.php`. Más específicamente, es un "arreglo asociativo" (esto es, Hash Table) con keys (claves) y valores. A propósito de ese formulario HTML en `hello.html`, una de tales claves debería ser `name`.

Pero, hablaremos más de eso en un rato.
Ahora, viene la parte divertida. Abrí `server.c`.
Sí, lo adivinaste. El reto que tenés en frente es implementar tu propio servidor web que conoce como "servir" contenido estático (esto es, archivos con extensión `.html`, `.jpg`, por ejemplo) y contenido dinámico (esto es, archivos terminando en `.php`).

5.1. server.c

Abrí `server.c`, si es que aún no lo has abierto. Tomemos un tour.

- En la parte superior del archivo hay un montón de "feature test macro requirements" que nos permiten usar ciertas funciones que son declaradas (condicionalmente) en archivos de encabezados mucho más abajo.
- Definidas a continuación hay ciertas constantes que especifican límites en los tamaños de las peticiones HTTP. Hemos (arbitrariamente) basado sus valores en configuraciones por defecto usadas por Apache, un servidor web popular. Mirá http://httpd.apache.org/docs/2.2/mod/core.html si tenés curiosidad.
- La siguiente definición es `BYTES`, una constante que especifica cuántos bytes estaremos eventualmente leyendo en buffers en un cierto tiempo.
- Luego, vienen un montón de cabeceras de archivos, seguidos por una definición de `BYTE`, la cual hemos de hecho definido como un `char` de 8-bits, seguido por un montón de prototipos.
- Finalmente, justo encima de main están algunas variables globales.

5.2. main

Ahora, demos una vuelta por `main`.

- En la parte superior de `main` hay una inicialización de lo que parece ser una variable global llamada `errno`. De hecho, `errno` está definida en `errno.h` y es usada por unas cuantas funciones para indicar (vía `int`), en casos de error, precisamente qué error ha ocurrido. Mirá `man errno` para más detalles.
- Un poco después de eso hay una llamada a `getopt`, la cual es una función declarada en `unistd.h` que hace más fácil revisar la escritura de argumentos de línea de comando. Mirá `man 3 getopt` si tenés curiosidad. Notá cómo usamos `getopt` (y algunas expresiones booleanas) para asegurarnos de que `server` es usado apropiadamente.

- Debajo de eso hay una declaración de una `struct sigaction` a través de la cual estaremos pendientes de `SIGINT` (esto es, control-c), llamando a `handler` (una función definida por nosotros en otro lado en `server.c`) si es escuchado control-c.
- Y luego, habiendo declarado algunas variables, `main` entra en un bucle `while` infinito.
    - En la parte superior del bucle, primero liberamos la memoria que pudo hber sido alocada por una interación previa del bucle.
    - Luego, revisamos si se ha precibido un control-c para detener el servidor.
    - Luego de eso, dentro de una declaración `if`, hay un llamado a `connected`, el cual retorna `true` si el cliente (por ejemplo, un navegador o incluso `curl`) se ha conectado al servidor.
    - Luego de eso hay un llamado a `parse`, la cual analiza una petición HTTP del navegador, almacenando su "abosolute path" (camino absoluto) y "query" (petición) dentro de dos arreglos que son pasados dentro de la función, por referencia.
    - Luego de eso, hay un montón de código que decodifica ese camino (decodificando caracteres codificados en una URL como `%20`) y "resuelve" el camino a un camino local, dándose cuenta exactamente de cuál archivo fue solicitado en el servidor mismo.
    - Debajo de eso, determinamos si el camino nos guía a un directorio o a un archivo y manemajamos la petición apropiadamente, en última instancia llamando a `list`, `interpret`, o `transfer`.
        - Para directorios (que no tienen un archivo `index.php` o `index.html` dentro de ellos), llamamos a `list` para mostrar el contenido del directorio.
        - Para archivos que terminan en `.php` (cuyos "MIME type" es `text/x-php`), llamamos a `interpret`.
        - Para otros archivos (supported), llamamos a `transfer`.

¡Y eso es todo para `main`! Notá, sin embargo, que a través de `main` hay algunos usos de `continue`, el efecto del cual es saltar al inicio del bucle infinito. En algunos casos, justo antes de `continue`, también, llamamos a `error` (otra función que escribimos) con un código status HTTP. Juntas, esas líneas permiten al servidor tratar y responder a errores antes de devolver su atención a nuevas solicitudes.

5.3. connected

Echá un vistazo rápido a `connected` debajo de `main`. Tranquilo si no sabés cómo opera esta función, pero deberías tratar de inferirlo de las págias de `man` para `memset` y `accept`.

5.4. error

Pasá un poco más de tiempo examinando `error`, la cual es aquella función a través la cual respondemos al navegador con errores (por ejemplo, 404). Esta función es un poquito más larga pero quizá tiene algunas contrucciones más familiares. Antes de seguir adelante, asegurate de estar razonablemente cómodo con cómo opera esta función. (Si tenés curiosidad, estamos usando `log10`) simplemente para conocer de cuántos dígitos, y entonces `char`s, tiene el código.

5.5. freedir

Esta función existe simplemente para facilitar la liberación de memoria que está alocada por una función llamada `scandir` que llamamos en `list`.

5.6. handler

¡Afortunadamente, una corta! Esta función (llamada cada vez que un usuario presiona control-c) esencialmente le dice a `main` que llame a `stop` al establecer `signaled`, una variable global, en `true`.

5.7. htmlspecialchars

Esta función, llamada idénticamente a esa función PHP que vimos más temprano, evacúa caracteres (por ejemplo, `<` como `<`) que podrían de otra forma "romper" una página HTML. Lo llamamos desde `list`, para que no se dé que algún archivo o directorio que estemos listando tenga un caracter "peligroso" en su nombre.

5.8. indexes

Ay, lo siento, nos olvidamos de implementar esta. Sobre eso...

5.9. interpret

Esta función habilita al servidor para interpretar archivos PHP. Es un poco críptico a primera vista, pero es pan comido. Todo lo que estamos haciendo es que al recibir una petición, digamos `hello.php`, está ejecutando una línea como

```
QUERY_STRING= "name=Alice" REDIRECT_STATUS=200 SCRIPT_FILENAME=/home/
ubuntu/workspace/pset6/public/hello.php php-cgi
```

el efecto de lo cual es pasar los contenidos de `hello.php` a un intérprete PHP (esto es `php-cgi`), con cualquier tipo de parámetro HTTP provistos a través de "environment variable" (variables ambientales) llama `QUERY_STRING`. A través de `load` (una función que escribimos), leemos el output del intérprete en la memoria (vía `load`). Y luego respondemos al navegador con (dinámicamente generado) una salida como:

```
HTTP/1.1 200 OK
X-Powered-By: PHP/5.5.9-1ubuntu4.12
Content-type: text/html

<!DOCTYPE html>

<html>
     <head>
         <title>hello</title>
     </head>
     <body>
         hello, Alice
     </body>
</html>
```

Incluso aunque el código PHP en `hello.php` se ve bonito impreso, su output no es tan lindo. (Echá un vistazo en `hello.php`. ¿Podés deducir por qué?)
Las apuestas son que `popen` no te será familiar. Esta función abre una "tubería"(pipe, en Inglés) a un proceso (`php-cgi` en nuestro caso), la cual nos provee de un puntero `FILE` a través de cual podemos leer el output estándar de ese proceso (como si fuera en la vida real).
Notá cómo esta función llama a `load` para leer el output del intérprete PHP en la memoria.

5.10. list

Ah, aquí hay una función que genera un listado de directorio. Notá cuánto código toma

generar HTML usando C, gracias a requisite memory management(no más sobre eso, ¡en PHP en Problem Set 7!)

5.11. load

Uf, una corta. Oh, esperá...

5.12. lookup

Rayos, otra vez.

5.13. parse

Y otro, al menos es el último de nuestros TO-DO's.

5.14. reason

Esta función simplemente proyecta los "status code" HTTP (por ejemplo, `200`) a "reason phrases" (por ejemplo, `OK`).

5.15. redirect

Esta función redirige un cliente a otra locación (esto es, URL) al enviar un código status de `301` más un encabezado `Location`.

5.16. request

Cuando el servidor recibe una petición de un cliente, el servidor no sabe a priori cuántos caracteres comprimirá la petición. Así, esta función lee iterativamente los bytes del cliente, un buffer a la vez, llamando `realloc` cuando sea necesario para almacenar el mensaje entero (esto es, request).
Notá el uso de esta función de punteros, alocación dinámica de memoria, aritmética de punteros, y más. Todo de alguna forma familiar para nosotros, pero definitivamente todo eso en un solo lugar. Intentá entender cada línea de código, al menos para practicar. En última instancia, se mantiene leyendo bytes para el cliente hasta que encuentre CRLF CRLF, lo cual, de acuerdo al spec HTTP, anuncia el final del encabezado de una petición. Si tenés curiosidad, `read` es algo parecido a `fread`, excepto que lee desde un "descriptor de archivo" (esto es, un `int`) en vez de desde un puntero `FILE`. Mirá su página `man` para más.

5.17. respond

Es esta función la que envía al cliente una respuesta HTTP, dado un código status, heads, un body, y la longitud de ese body. Por ejemplo, es esta función la que envía una respuesta como la siguiente.

```
HTTP/1.1 200 OK
X-Powered-By: PHP/5.5.9-1ubuntu4.12
Content-type: text/html

<!DOCTYPE html>

<html>
     <head>
         <title>hello</title>
     </head>
     <body>
         hello, Alice
     </body>
</html>
```

Por cierto, `dprintf` es como `printf` (o, en realidad, `fprintf`) excepto por que el último, como `read`, escribe a un "descriptor de archivo" en vez de hacerlo a un `FILE*`.

5.18. start

Aquí está la función que lo inició todo. No te preocupés si (incluso con `man`) no entendés todas sus líneas, particularmente el código de networking. Pero tené en mente que `start` es la función que configura al servidor para que escuche conexiones en un puerto TCP particular.

5.19. stop

Y `stop` hace justo lo opuesto, liberando toda la memoria y en última instancia obligando al servidor a salir, incluso sin devolver el contol a `main`.

5.20. transfer

El propósito de esta función en la vida es transferir un archivo desde el servidor a un cliente. Donde `interpret` maneja contenido dinámico (generado por scripts PHP),

`transger` trata con contenido estático (ejemplo, JPEGs). Notá cómo esta función llama a `load` para leer cierto archivo del disco

5.21. urlcode

Esta función, también nombrada después de una función PHP, decidifica en URL una string, convirtiendo caracteres especiales como `%20` de vuelta a sus valores originales.

6. What To Do

Bien, ataquemos esos `TO-DO`s

https://www.youtube.com/watch?v=BYdgkUkchbQ

6.1. lookup

Completá la implementación de `lookup` en tal forma que retorne

- `text/css` para cualquier archivo cuyo `path` termine en `.css` (o cualquier combinación de mayúsculas y minúsculas),
- `text/html` para cualquier archivo cuyo `path` termine en `.html` (o cualquier combinación de mayúsculas y minúsculas),
- `image/gif` para cualquier archivo cuyo `path` termine en `.gif` (o cualquier combinación de mayúsculas y minúsculas),
- `image/x-icon` para cualquier archivo cuyo `path` termine en `.ico` (o cualquier combinación de mayúsculas y minúsculas),
- `image/jpeg` (no `image/jpg`) para cualquier archivo cuyo `path` termine en `.jpg` (o cualquier combinación de mayúsculas y minúsculas),
- `text/javascript` para cualquier archivo cuyo `path` termine en `.js` (o cualquier combinación de mayúsculas y minúsculas),
- `text/x-php` para cualquier archivo cuyo `path` termine en `.php` (o cualquier combinación de mayúsculas y minúsculas),
- `image/png` para cualquier archivo cuyo `path` termine en `.png` (o cualquier combinación de mayúsculas y minúsculas),
- `text/css` para cualquier archivo cuyo `path` termine en `.css` (o cualquier combinación de mayúsculas y minúsculas),

- `NULL` si no es ninguno de los casos anteriores.

Apostamos a que encontrarás útiles funciones como `strcasecmp`, `strcpy`, y/o `strrchr`.

6.2. parse

Completá la implementación de `parse` de tal forma que la función "parsea" (analiza) (itera sobre) `line`, extrayendo su absolute-path (camino-abosuluto) y su query guardándolos en `abspath` y `query` (consulta), respectivamente.
Aquí está cómo.
`abs_path`
Por 3.1.1 de https://tools.ietf.org/html/rfc7230, una `request-line` está definida como

```
method SP request-target SP HTTP-version CRLF
```

donde `SP` representa un solo espcio ( ) y `CRLF` representa `\r\n`. Mientras tanto, ni `method`, `request-target`, ni `HTTP-version` pueden contener a `SP`.
Por 5.3 del mismo RFC, `request-target`, sin embargo, puede tomar varias formas, de las cuales, la única que tu servidor necesita sostener es

```
absolute-path [ "?" query ]
```

donde `absolute-path` (el cual no contendrá `?` debe iniciar con `/` y puede ser opcionalmente seguido por un `?` seguido por una `query`, la cual puede no contener `"`
Asegurate que `request-line` (el cual es pasado en `parse` como `line`) es consistente con estas reglas. Si no lo es, respondé al navegador con **404 Bad Request** y retorná `false`.
Incluso si `requet-line` es consitente con estas reglas,

- si `method` no es `GET`, respondé al buscador con **405 Method Not Allowed** y retorná `false`;
- Si `requested-target` no comienza con `/`, respondé al navegador con **501 Not Implemented** y retorná `false`;
- Si `request-target` contiene un `"`, respondé al navegador con **404 Bad Request** y retorná `false`;
- Si `HTTP-version` no es `HTTP/1.1`, respondé al navegador con **505 HTTP Version Not Supported** y retorná `false`; o

Apostamos a que funciones como `strchr`, `strcpy`, `strncmp`, `strncpy`, y/o `strstr` te serán útiles.
Si todo va bien, almacena `absolute-path` en las direcciones en `abs_path` (el cuál también fue también en `parse` como un argumento). Podés asumir que la memoria a la que `abs_path` apunta sería de longitud de al menos `LimitRequestLine` + 1.

`query`

Almacena en las direcciones en `query` la substring (subcadena) `query` provenientes de `request-target`. Si esa substring está ausente (incluso si un `?` está presente), entonces `query` debería ser `""`, comsumiento de este modo un byte, donde `query[0]` es `'\0'`. Podés asumir que la memoria a la que apunta `query` será de longitud al menos `LimitRequestLine` + 1.
Por ejemplo, si `request-target` es `/hello.php` o `/hello.php?`, entonces `query` debería tener un valor de `""`. Y si `request-target` is `/hello.php?q=Alice`, entonces `query` debería tener un valor de `q=Alice`. De seguro notarás la utilidad de funciones como `strchr`, `strcpy`, `strncpy`, y/o `strstr`.

6.3. load

Completá la implementción de `load` de forma tal que la función:

1. lea todos los bytes disponibles de `file`,
2. almacena esos bytes de forma contigua en memomoria dinámicamente alocada en cola (heap),
3. almacene las direccione del primero de esos bytes en `*content`, y
4. almacene el número de bytes en `*length`.

Notá que `content` es un "puntero a un puntero" (eso es, `BYTE**`), lo cual significa que podés efectivamente "retornar" un `BYTE*` a cualquier función llamada por `load` al dereferenciar `content` y al almacenar la direción de un `BYTE` en `*content`. Mientras tanto, `length` es un puntero (esto es, `size_t*`, el cual podés además dereferenciar para que "retorne" un `size_t` a cuál sea la función llamada por `load` al dereferenciar `length` y almacenar un número en `length`.

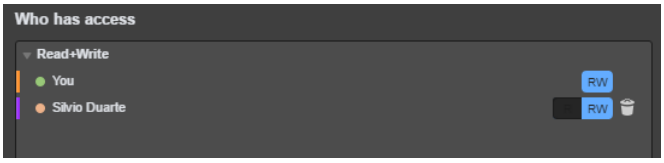
6.4. indexes

Completá la implementación de `indexes` de tal forma que la función, dado un `/path/to/a/directory`, retorne `/path/to/a/directory/index.php` si `index.php` en realidad existe ahí, o `/path/to/a/directory/index.html` si `index.html` en realidad existe ahí, o `NULL`. En el primero de esos casos, esta función debería alocar memoria dinámicamente en la cola (heap) para la string retornada.

7. Entrega

Debe de compartir su espacio de trabajo en su cuenta de `cs50.io` por lo cual deberá de seguir los siguientes pasos (por favor omitir los que muestran cómo crear la cuenta desde cero si ya tiene una. Estas instrucciones ya estaban descritas también desde el PSet1: `Crypto` y si ya compartió su espacio de trabajo no es necesario repetir el proceso):

- Acceda a edx.org y dé clic en el botón Registrer (parte superior derecha del sitio).
- Rellene todos los datos que se le solicitan o acceda con una de sus redes sociales (es preferencial).
- Una vez que se registró diríjase a cs50.io y será redirigido a una página donde deberá escoger la opción **edX** como **CS50 ID** (dé clic en **Submit**).
- Se le redirigirá a un formulario de acceso del sitio de **edx.org** donde deberá ingresar su correo electrónico y contraseña.
- Una vez que ha accedido ya podrá compartir su **Workspace** dando clic en el botón **Share** ubicado en la **parte superior derecha.**
- Dentro del mismo formulario en la sección **Invite People** escriba **silfdv** y proporcione permisos de escritura (botón **RW**), luego de clic en **Invite**, le debería aparecer lo siguiente:



- Una vez que creó sus cuentas en **edX** y en el **IDE de CS50**, al mismo tiempo que compartió el **Workspace** con el usuario **silfdv** ya podremos revisar su código.

Esto fue Pset6.