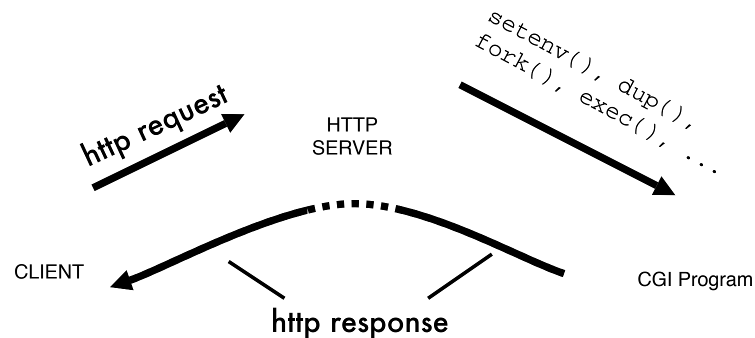
CSC 498: Web Programming

Haidar Harmanani
Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010 Lebanon

Common Gateway Interface

- CGI is a standard mechanism for:
  - Associating URLs with programs that can be run by a web server.
  - A protocol (of sorts) for how the request is passed to the external program.
  - How the external program sends the response to the client.
- Simply stated
  - CGI is an interface between HTML forms and server-side scripts

CGI Programming



Simple Example

```
<form action="http://www.csc.byblos.lau.edu.lb/cgi-bin/run/~haidar/mult.cgi">
<div><label>Multiplicand 1: <input name="m" size="5"></label></div>
<div><label>Multiplicand 2: <input name="n" size="5"></label></div>
<div><input type="submit" value="Multiply!"></div>
</form>
```

Multiplicand 1:
Multiplicand 2:
Multiply!

Analyzing the Example

- Assume that you type 4 into one input field and 9 into another and then you click the submit button
  - The browser will send, by the HTTP protocol, a request to the server at www.csc.byblos.lau.edu.lb
    - The browser pick up this server name from the value of ACTION attribute where it occurs as the host name part of a URL
  - When sending the request, the browser provides additional information, specifying a relative URL, in this case

    /cgi-bin/run/~jkorpela/mult.cgi?m=4&n=9
- This was constructed from that part of the ACTION value that follows the host name, by appending a question mark "?" and the form data in a specifically encoded format.
- The server to which the request was sent will then process it according to its own rules.
- The server invokes a script or a program specified in the URL (mult.cgi in this case) and passes some data to it (the data m=4&n=9 in this case).

Now, in more details...

CGI URLs

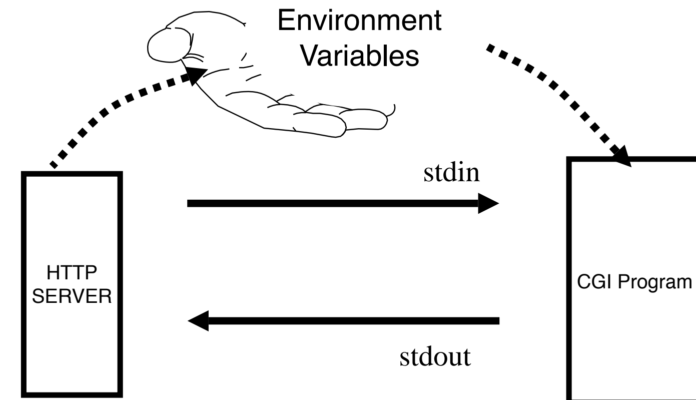
- There is some mapping between URLs and CGI programs provided by a web sever
- The exact mapping is not standardized (web server admin can set it up).
- Typically:
  - requests that start with `/CGI-BIN/` , `/cgi-bin/` or `/cgi/`, etc. refer to CGI programs (not to static documents).

Request ⟶ CGI program

- The web server sets some environment variables with information about the request.
- The web server fork()s and the child process exec()s the CGI program.
- The CGI program gets information about the request from environment variables.

STDIN, STDOUT

- Before calling exec(), the child process sets up pipes so that stdin comes from the web server and stdout goes to the web server.
- In some cases part of the request is read from stdin.
- Anything written to stdout is forwarded by the web server to the client.



Important CGI Environment Variables

- REQUEST_METHOD
- QUERY_STRING
- CONTENT_LENGTH

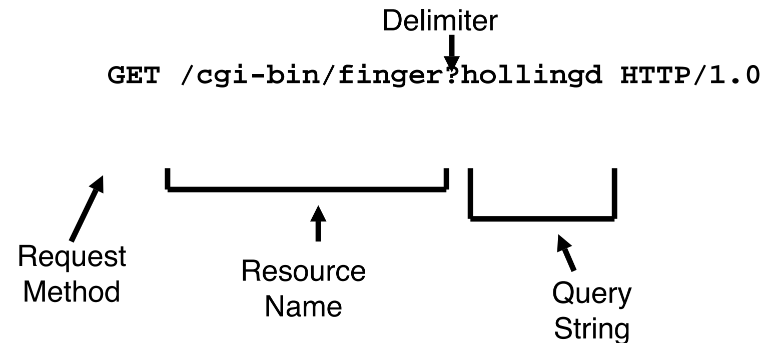
> **Environment variables are a set of dynamic <u>values that can affect the way running processes will behave on a Computer</u>**
>
> *Wikipedia*

Request Method: Get

- GET requests can include a *query string* as part of the URL:

Delimiter

```
GET /cgi-bin/finger?hollingd HTTP/1.0
```



/cgi-bin/finger?hollingd

- The web server treats everything before the '?' delimiter as the resource name
- In this case the resource name is the name of a program.
- Everything after the '?' is a string that is passed to the CGI program.

Simple GET queries - ISINDEX

- You can put an <ISINDEX> tag inside an HTML document.
- The browser will create a text box that allows the user to enter a single string.
- If an ACTION is specified in the ISINDEX tag, when the user presses Enter, a request will be sent to the server specified as the ACTION.

ISINDEX Example

```
Enter a string:
<ISINDEX ACTION=http://foo.com/search.cgi>
Press Enter to submit your query.
```

If you enter the string "blahblah", the browser will send a request to the http server at `foo.com` that looks like this:

```
GET /search.cgi?blahblah HTTP/1.1
```

What the CGI sees

- The CGI Program gets `REQUEST_METHOD` using `getenv`:

```
char *method;
method = getenv("REQUEST_METHOD");
if (method==NULL) … /* error! */
```

Getting the GET

- If the request method is GET:

```
if (strcasecmp(method,"get")==0)
```

- The next step is to get the query string from the environment variable `QUERY_STRING`

```
char *query;
query = getenv("QUERY_STRING");
```

Send back http Response and Headers:

- The CGI program can send back a http status line :

```
printf("HTTP/1.1 200 OK\r\n");
```

- and headers:

```
printf("Content-type: text/html\r\n");
printf("\r\n");
```

Important!

- A CGI program doesn't have to send a status line (the http server will do this for you if you don't).
- A CGI program must always send back at least one header line indicating the data type of the content (usually text/html).
- The web server will typically throw in a few header lines of it's own (Date, Server, Connection).

Simple GET handler

```
int main() {
  char *method, *query;
  method = getenv("REQUEST_METHOD");
  if (method==NULL) … /* error! */
  query = getenv("QUERY_STRING");
  printf("Content-type: text/html\r\n\r\n");
  printf("<H1>Your query was %s</H1>\n",
          query);
 return(0);
}
```

URL-encoding

- Browsers use an encoding when sending query strings that include special characters.
  - Most nonalphanumeric characters are encoded as a `'%'` followed by 2 ASCII encoded hex digits.
  - `'='` (which is hex 3D) becomes `"%3D"`
  - `'&'` becomes `"%26"`

More URL encoding

- The space character `' '` is replaced by `'+'`.
  - Why? (think about project 2 parsing…)
- The `'+'` character is replaced by **`"%2B"`**

Example:

`"foo=6 + 7"` becomes **`"foo%3D6+%2B+7"`**

Security!!!

- It is a ***very*** bad idea to build a command line containing user input!
- What if the user submits: **`" ; rm -r *;"`**

```
grep ; rm -r *; /usr/dict/words
```

Beyond ISINDEX - Forms

- Many Web services require more than a simple ISINDEX.
- HTML includes support for forms [see lecture 3]:
  - lots of field types
  - user answers all kinds of annoying questions
  - entire contents of form must be stuck together and put in QUERY_STRING by the Web server.

Form Fields

- Each field within a form has a name and a value.
- The browser creates a query that includes a sequence of "name=value" substrings and sticks them together separated by the '&' character.

Form fields and encoding

- 2 fields - name and occupation.
- If user types in "Dave H." as the name and "none" for occupation, the query would look like this:

```
"name=Dave+H%2E&occupation=none"
```

HTML Forms

- Each form includes a METHOD that determines what http method is used to submit the request.
- Each form includes an ACTION that determines where the request is made.

An HTML Form

```
<FORM METHOD=GET ACTION=http://foo.com/
 signup.cgi>
Name:
<INPUT TYPE=TEXT NAME=name><BR>
Occupation:
<INPUT TYPE=TEXT NAME=occupation><BR>
<INPUT TYPE=SUBMIT>
</FORM>
```

What a CGI will get

- The query (from the environment variable QUERY_STRING) will be a URL-encoded string containing the name,value pairs of all form fields.
- The CGI must decode the query and separate the individual fields.

HTTP Method: POST

- The HTTP POST method delivers data from the browser as the content of the request.
- The GET method delivers data (query) as part of the URI.

GET vs. POST [Recap]

- When using forms it's generally better to use POST:
  - there are limits on the maximum size of a GET query string (environment variable)
  - a post query string doesn't show up in the browser as part of the current URL.

HTML Form using POST

Set the form method to POST instead of GET.

```
<FORM METHOD=POST ACTION=…>
```

The browser will take care of the details…

CGI reading POST

- If REQUEST_METHOD is a POST, the query is coming in STDIN.
- The environment variable CONTENT_LENGTH tells us how much data to read.

CGI Method summary

- GET:
  - **`REQUEST_METHOD`** is "GET"
  - **`QUERY_STRING`** is the query
- POST:
  - **`REQUEST_METHOD`** is "POST"
  - **`CONTENT_LENGTH`** is the size of the query (in bytes)
  - query can be read from STDIN

More about CGI

- Keeping track of state information.
- Cookies.
- Image Mapping
- Authentication

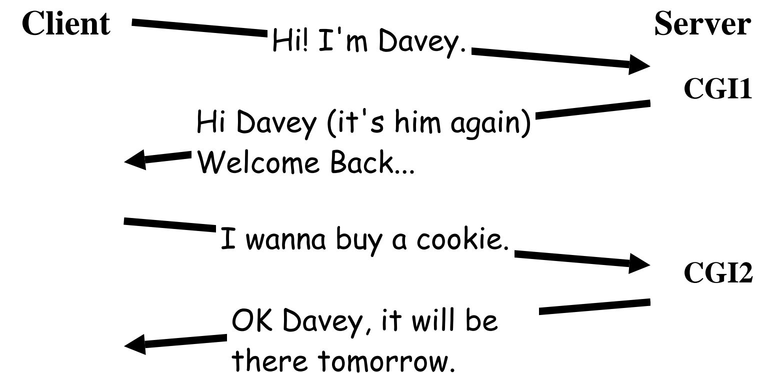
Sessions

- Many web sites allow you to establish a session.
  - you identify yourself to the *system.*
  - now you can visit lots of pages, add stuff to shopping cart, establish preferences, etc.

State Information

- Remember that each HTTP request is unrelated to any other (as far as the Web server is concerned).
- Each new request to a CGI program starts up a brand new copy of the CGI program.
- Providing sessions requires keeping state information.

Session Conversation



Hidden Field Usage

- One way to propogate state information is to use hidden fields.
- User identifies themselves to a CGI program (fills out a form).
- CGI sends back a form that contains hidden fields that identify the user or session.

Revised Conversation

Initial form has field for user name.

```
GET /cgi1?name=davey HTTP/1.0
```

CGI1 creates order form with hidden field.

```
GET/cgi2?name=davey&order=cookie HTTP/1.0
```

```
<!-- The login form - notice the hidden field formname -->
<CENTER>
<H2 STYLE="font-family: Helvetica;
        color: Blue;
                font-weight: bold;
        font-size: 14pt">
Please enter your name and password
</H2><P>

<FORM METHOD=POST ACTION=pizza>
<INPUT TYPE=HIDDEN NAME=formname VALUE=login>
<TABLE>
<TR>
  <TD  ALIGN=RIGHT
     STYLE="font-family: Helvetica;
          color: Purple; font-weight: bold; font-size: 14pt">
   Name: </TD>
  <TD> <INPUT TYPE=TEXT NAME="name" WIDTH=10></TD>
</TR>
<TR>
  <TD  ALIGN=RIGHT
     STYLE="font-family: Helvetica; color: Purple; font-weight: bold;
          font-size: 14pt">
  Password:
  </TD>
  <TD> <INPUT TYPE=PASSWORD NAME="password" WIDTH=10></TD>
</TR>
<TR ALIGN=CENTER>
  <TD COLSPAN=2> <INPUT TYPE=SUBMIT Value="Login"></TD>
</TR>
</TABLE>
</CENTER>
</FORM>
```

Session Keys

- Many Web based systems use hidden fields that identify a session.
- When the first request arrives, the system generates a unique session key and stores it in a database.
- The session key can be included in all forms/links generated by the system (as a hidden field or embedded in a link).

Session Key Properties

- Must be unique.
- Should expire after a while.
- Should be difficult to predict.
  - typically use a pseudo-random number generator seeded carefully.

Pizza Server Session Keys

- We could change the pizza server system to use session keys:

```
<INPUT TYPE=HIDDEN NAME=sessionkey
 VALUE=HungryStudent971890237>
```

Pizza Order

- A request to order a pizza might now look like this (all on one line):

```
GET /pizza.cgi?sessionkey=
HungryStudent971890237&pizza=cheese&
size=large  HTTP/1.0
```

Cookies

- Create by Netscape as part of Netscape extensions to CGI
- The way cookies work is as follows:
  - The web server give "Set-Cookie:" header with information on cookie
  - The browser keep track of cookie.
  - When a browser accesses a page on the same site, it sends back a list of all cookies for that site

HTTP Cookies

- Technically, a "cookie" is a (name, value) pair that a CGI program can ask the client to remember.
- The client sends this (name, value) pair along with every request to the CGI.
- We can also use "cookies" to propagate state information.

Cookies are HTTP headers

- Cookies are HTTP headers.
- A server (CGI) can give the browser a cookie by sending a Set-Cookie header line with the response.
- A client can send back a cookie by sending a Cookie header line with the request.

Setting a cookie

```
HTTP/1.0 200 OK
Content-Type: text/html
Set-Cookie: customerid=0192825
Content-Length: 12345
Favorite-Cookie: Choco-Chip
Nap-Time: 12-2
...
```

`Set-Cookie` Header Options

The general form of the Set-Cookie header is:

`Set-Cookie: name=value;` ***`options`***

The options include:

`expires=...`

`domain=...`

`path=...`

`expires` Option

```
expires=Friday 29-Feb-2000 00:00:00 GMT
```

- This tells the browser how long to hang on to the cookie.
- The time/date format is very specific!

`expires` Time Format

```
Weekday, Day-Month-Year
 Hour:Minute:Second GMT
```

- This all must be on one line!
- Weekday is spelled out.
- Month is 3 letter abbreviation
- Year is 4 digits

Default expiration

- If there is no expires option on the **`Set-Cookie`** header line, the browser does not save the cookie to disk.
- In this case, when the browser is closed it will forget about the cookie.

`domain` Option

`domain=.rpi.edu`

- The domain option tells the browser the *domain(s)* to which it should send the cookie.
- *Domains* as in DNS.
- The domain must start with "." and contain at least one additional "."

`Domain` option rules

- The server that sends the Set-Cookie header must be in the domain specified.
- If no domain option is in the header, the cookie will only be sent to the same server.

Default Behavior

`path` Option

`path=/`

or

`path=/~hollingd/netprog`

- The path option tells the browser what URLs the cookie should be sent to.

`path` default

- If no path is specified in the header, the cookie is sent to only those URLs that have the same path as the URL that set the cookie.
- A path is the leading part of the URL (does not include the filename).

Default Path Example

If the cookie is sent from:

```
/~hollingd/netprog/pizza/pizza.cgi
```

it would also be sent to

```
/~hollingd/netprog/pizza/blah.cgi
```

but not to

```
/~hollingd/netprog/soda/pizza.cgi
```

Set-Cookie Fields

- Many options can be specified.
- Things are separated by ";":

```
Set-Cookie: a=blah; path=/;
 domain=.cs.rpi.edu; expires=Thursday,
 21-Feb-2002 12:41:07 2002
```

All must be on one line!

CGI cookie creation

- A CGI program can send back any number of HTTP headers.
  - can set multiple cookies
- Content-Type is required!
- Blank line ends the headers!

C Example

```
printf("Content-Type: text/html\r\n");
printf("Set-Cookie: prefs=nofrms\r\n");
printf("Set-Cookie: Java=yes\r\n");
printf("\r\n");
```

… now sends document content

Getting HTTP Cookies

- The browser sends each cookie as a header:

```
Cookie: prefs=nofrms
Cookie: Java=OK
```

- The Web server gives the cookies to the CGI program via an environment variable.

Multiple Cookies

- There can be more than one cookie.
- The Web Server puts them all together like this:

```
prefs=nofrms; Java=OK
```

maybe a space, maybe not!

and puts this string in the environment variable: **HTTP_COOKIE**

Cookie Limits

- Each cookie can be up to 4k bytes.
- One "site" can store up to 20 cookies on a user's machine.

Cookie Usage

- Create a session.
- Track user browsing behavior.
- Keep track of user preferences.
- Avoid logins.

Cookies and Privacy

- Cookies can't be used to:
  - send personal information to a web server without the user knowing about it.
  - be used to send viruses to a browser.
  - find out what other web sites a user has visited.*
  - access a user's hard disk

* although they can come pretty close to this one!

Some Issues

- Persistent cookies take up space on user's hard disk.
- Can be used to track your behavior within a web site.
  - This information can be sold or shared.
- Cookies can be shared by cooperating sites (advertising agencies do this).

Cookie Examples

- showcookie.cgi
  - sends back an HTML table that contains a list of the cookies sent.
  - also sends a form that tells the CGI what cookie we would like it to set.
- pizzacookie
  - pizza server that uses a cookie for propogating state information.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <ctype.h>

typedef struct {
	char *key;
	char *val;
} Cookie;

char *CopyKey(char *);
char *CopyVal(char *);

static char *favorite_cookies[] = {
	"chocolate",
	"vanilla",
	"sprinkles",
	"HTTP",
	NULL
};
```

```
Cookie *ParseCookie(char *env, int *n)
{
	Cookie *c = NULL;
	int nc = 0;
	char *cookiestr = NULL;

	if (!env)
	{
		*n = 0;
		return NULL;
	}

	/* add ; to end of string for easier parsing */
	cookiestr = malloc(strlen(env) + 2);
	sprintf(cookiestr, "%s;", cookiestr);
...
```



```
env = cookiestr;
	while (*env)
	{
		char *next = strstr(env, ";");
		if (next)
		{
			/* Get this token */
			c = realloc(c, sizeof(Cookie)*(nc + 1));
			c[nc].key = CopyKey(env);
			c[nc].val = CopyVal(env);
			nc++;

			/* find next token */
			env = next + 1;
			while (*env && isspace(*env))
				env++;
		}
	}
	free(cookiestr);
	*n = nc;
	return c;
}
```



```
char *CopyKey(char *pair)
{
	char *p = strchr(pair, '=');
	char *ret = NULL;
	if (p)
	{
		int len = p - pair;
		ret = malloc(len + 1);
		memcpy(ret, pair, len);
		ret[len] = '\0';
	}
	return ret;
}
```

```
char *CopyVal(char *pair)
{
	char *ret = NULL;
	int len;
	pair = strchr(pair, '=');
	if (pair)
	{
		pair++;
		len = strchr(pair, ';') - pair;
		ret = malloc(len + 1);
		strncpy(ret, pair, len);
		ret[len] = '\0';
	}

	return ret;
}
```



```
int main(void)
{
	char expirestr[200];
	Cookie *cookies;
	int num_cookies;
	int i;

	/* We format page in HTML */
	printf("Content-type: text/html\n");

	/* If we have parameter, set cookie */
	if (getenv("QUERY_STRING"))
	{
		/* we make expire in one hour */
		time_t expires = time(NULL) + 3600;

		/* format string can be "%z" in GNU libc */
		strftime(expirestr, 200, "%a,  %d  %b  %Y  %H:%M:%S  %z",
			gmtime(&expires));
```

```
for (i = 0; i < num_cookies; i++)
            printf("<tr><td>%s</td><td>%s</td></tr>\n",
                cookies[i].key,
                cookies[i].val);

        printf("</table></p>\n");
    }

    printf("<P>What is you favorite kind of cookie?<ul>\n");

    for (i = 0; favorite_cookies[i]; i++)
        printf ("<li><a href=\"?%s\">%s</a></li>\n",
            favorite_cookies[i],
            favorite_cookies[i]);

    printf("</ul></p></body></html>");

    return 0;
}
```

```
/* Set cookie! */
        printf("Set-Cookie: FAVORITE=%s; Expires=%s\n",
            getenv("QUERY_STRING"),
            ctime(&expires));
    }

    printf("\n");

    printf("<html><head><title>CGI is for Cookie</title></head><body>\n");

    if (getenv("HTTP_COOKIE"))
    {
        printf("<P>Hello children and furry creature!  You have these "
            "cookie.  GIVE ME COOKIE!\n");
        printf("<table border=2>");
        printf("<tr><th>Name</th><th>Value</th></tr> \n");

        cookies = ParseCookie(getenv("HTTP_COOKIE"), &num_cookies);
```