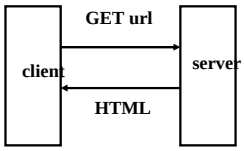
HTTP: Hypertext transfer protocol

- **What happens when you click on a URL?**
- **client sends request:**

  `GET` ***`url`*** `HTTP/1.0`

  `(blank line)`



- **server returns**

  `header info`

  `(blank line)`

  `HTML`
  - since server returns the text, it can be created as needed
  - can contain encoded material of many different types (MIME)

- **URL format**

  `service://hostname/filename?other_stuff`

`filename?other_stuff` **part can encode**
- data values from client (forms)
- request to run a program on server (cgi-bin)

Embellishments

- **basic design just returns text to be displayed**

- **helpers or plug-ins to display non-text content**
  - pictures (GIF, JPEG), sound, video, …

- **forms filled in by user**
  - client encodes form information in URL or on stdout
  - server interprets it from environment or stdin
    - usually with cgi-bin program
    - can be written in anything: Perl, PHP, shell, C, Java, …

- **HTTP is stateless**
  - server doesn't save anything from one request to next
  - need a way to remember information on the client
    - cookies

- **active content: download code to run on client**
  - Javascript and other interpreters
  - Java applets
  - plug-ins
  - ActiveX

Forms and CGI-bin programs

- **"common gateway interface"**
  - standard way to ask the server to run a program
  - using information provided by the client
  - usually via a form
- **if target file on server is executable program,**
  - e.g., in /cgi-bin directory
- **and if it has right permissions, etc.,**

- **server runs it to produce HTML to send to client**
  - using the contents of the form as input

- **CGI programs can be written in any language**
  - Perl, PHP, C, shell, ASP, JSP, …

- **CGI facility: campuscgi.princeton.edu**
  - anyone can run CGI scripts
  - restrictions on what scripts can access and what they can do

HTML form hello.html

```
<html>
<body>

<FORM ACTION=
  "http://campuscgi.princeton.edu/~bwk/hello1.cgi"
       METHOD=GET>
<INPUT TYPE="submit"
        value="hello1: shell script, plain text">
</FORM>

<FORM ACTION=
  "http://campuscgi.princeton.edu/~bwk/hello2.cgi"
       METHOD=GET>
<INPUT TYPE="submit"
        value="hello2: shell script, html">
</FORM>

</body>
</html>
```

Simple echo scripts hello[12].cgi

- **Plain text… (hello1.cgi)**

```
#!/bin/sh
echo "Content-type: Text/plain"
echo
echo Hello, world.
```

- **HTML … (hello2.cgi)**

```
#!/bin/sh
echo 'Content-Type: text/html

<html>
<title> Hello2 </title>
<body bgcolor=cyan>
<h1> Hello, world </h1>'

echo "<h2> It's `date` </h2>"
```

- **These have no user input or parameters**
- **though content can change (as in hello2)**

Dynamically created content

- **using Perl (hello3.cgi)**
- **`…` executes command, returns result as string**
- **<<str … str quotes contents,**
  - with interpolation of $var, `…`, etc.
  - terminating str has to be on a line by itself
  - "here document" in Bourne shell terminology

```
#!/usr/princeton/bin/perl

$date = `/bin/date`;
print <<END;
Content-Type: text/html

<html>
<title> Hello3 </title>
<body bgcolor=yellow>
<h1> Hello, world </h1>

<h2> It's $date </h2>
END
```

HTML forms: data from users (surv0.html)

```
<html>
<title> COS 333 Survey </title>
<body> <h2> COS 333 Survey </h2>
<form METHOD=GET ACTION=
    "http://campuscgi.princeton.edu/~bwk/surv0.cgi">
<br> Name: <input type=text name=Name size=40>
<br> Class: <input type=radio name=Class value=06> '06
<br> Class: <input type=radio name=Class value=07> '07
<P> CS courses:
<input type=checkbox name=c126> 126
<input type=checkbox name=c217> 217
<P> Experience?
<textarea name=exper rows=3 cols=40 wrap>
</textarea>
<P>
<input type=submit> <input type=reset>
</form>
Thanks.
</body></html>
```

Retrieving info from forms (server side)

- HTTP server passes info to your cgi program in environment variables
- form data available in environment variable QUERY_STRING (GET) or on stdin (POST)

- campuscgi.princeton.edu/~bwk/surv0.cgi :

```
foreach $i (sort keys %ENV) {
        $env .= "<br> $i $ENV{$i}";
}

print <<END;
Content-Type: text/html

<html>
<body bgcolor=white>
<h3>
query = $ENV{"QUERY_STRING"}
<p>
env = $env
</h3>
END
```

URL encoding of form data

- **how form data gets from client to server**
  - http://hostname/restofpotentiallyverylongline
  - everything after hostname interpreted by server
  - usually /program?encoded_arguments
- **if form uses GET, encoded in URL format in QUERY_STRING environment variable**
- **if form uses POST, encoded in URL format on stdin (CONTENT_LENGTH bytes)**
- **URL format:**
  - keywords in keyword lists separated by +
  - parameters sent as name=value&name=value
  - funny characters encoded as %NN (hex)
  - you have to parse the string; it's a mess

```
# surv0a.html, surv0a.cgi
read(STDIN, $q, $ENV{CONTENT_LENGTH});
print <<END;
Content-Type: text/html

<html>
<body>
query = $q
END
```

Defensive programming

```
char postString[1024];

contentLength =
    atoi(getenv("CONTENT_LENGTH"));
cin.read(postString,contentLength);
```

from a C++ book (4th edition, 2003)

- **program defensively**

  "Always validate all your inputs -- the world outside your function should be treated as hostile and bent upon your destruction."

Howard & LeBlanc, *Writing Secure Code*, p 80

Extracting URL data by brute force

- **surv1.cgi**:

```
my %params;

read(STDIN, $q, $ENV{CONTENT_LENGTH});
parse($q);
foreach $i (sort keys %params) {
        $s .= "$i = $params{$i}<br>\n";
}

print <<END;
Content-Type: text/html

<html>
<body bgcolor=white>
<h3>
query = $q
<p>
params = $s
END
```

(continued on next page)

Brute force, part 2

```
sub parse {
    my $temp = "@_";
    my @pairs = split('&', $temp);
    my($par, $val);

    foreach (@pairs) {
        ($par, $val) = split('=');
        $par = unescape($par);
        $val = unescape($val);
        if ($params{$par}) {
            $params{$par} .= "$;$val";
        } else {
            $params{$par} = $val;
        }
    }
}

sub unescape {
    my $temp = "@_";
    $temp =~ tr/+/ /;  # translate + to space
    $temp =~ s/%([0-9a-fA-F]{2})/
                        pack("c",hex($1))/ge;
    return $temp;
}
```

Perl CGI.pm package (surv2.cgi)

- **parses URL data, generates HTML**

```
use CGI;
$query = new CGI;

print $query->header;
print $query->start_html(-title=>'CS 333
  Survey', -bgcolor=>'white');
print "<h1> CS 333 Survey </h1>\n";

print "<P>\n";
foreach $name ($query->param) {
        $value = $query->param($name);
        $s = $s . $name . " " . $value . "\n";
        print "<br> $name $value\n";
}
$s .= "Host " . $query->remote_host();
$s .= " " . $query->remote_addr();
print "<P> $s\n";
print $query->end_html();

open(MAIL, "|mail bwk");
print MAIL "$s\n";
close MAIL;
```

PHP (www.php.com)

- **an alternative to Perl for Web pages**
- **sort of like Perl turned inside-out**
  - text sent by server
  - after PHP within it has been executed

- **hello.php:**

```
<html>
<title> PHP hello</title>
<body bgcolor=lightyellow>

<h2> Hello from PHP </h2>
<?php
echo "It's " . date("F j, Y, g:i a");
echo "<P>";
?>

</body>
</html>
```

PHP version of survey (survey.php)

```
<html>
<title>COS 333 Survey</title>
<h4> COS 333 Survey</h4>

<?php
echo "ENV====\n";
foreach ($_ENV as $key => $value) {
        echo "<br> $key = $value\n";
}
echo "POST=====\n";
$s = "";
foreach ($_POST as $key => $value) {
        echo "<br> $key = $value\n";
        $s .= "$key = $value\n";
}
echo "SERVER======\n";
foreach ($_SERVER as $key => $value) {
        echo "<br> $key = $value\n";
}
?>
<P>
<?php $b = mail("bwk", "survey reply", $s);
echo "mail status = $b\n";
echo "mail message = [$s]\n";
?>

</body>
</html>
```

Why scripting languages?

- very expressive
- efficient enough
- extensible (usually)
- portable
- reliable

- good for glue, prototyping,
- sometimes good for production

- see Ousterhout's scripting paper on web page