

# In the good old days...

- Years ago... the WWW was made up of (mostly) static documents.
  - Each URL corresponded to a single file stored on some hard disk.
- Today - many of the documents on the WWW are built at request time.
  - URL doesn't correspond to a single file.

# Dynamic Documents

- Dynamic Documents can provide:
  - automation of web site maintenance
  - customized advertising
  - database access
  - shopping carts
  - date and time service
  - well paying jobs for netprog students.

**Cool idea!**



# Web Programming

- Writing programs that create dynamic documents has become very important.
- There are a number of general approaches:
  - Create custom server for each service desired.
    - Each is available on different port.
  - Have web server run external programs.
  - Develop a real smart web server
    - SSI, scripting, server APIs.

# Custom Server

- Write a TCP server that watches a “well known” port for requests.
- Develop a mapping from http requests to service requests.
- Send back HTML (or whatever) that is created/selected by the server process.
- Have to handle http errors, headers, etc.

# An Example Custom Server

- We want to provide a time and date service.
- Anyone in the world can find out the date and time (according to our computer)!!!
- We don't care what is in the http request, our reply doesn't depend on it.
- We assume the request comes from a browser that wants the content formatted as an HTML document.

# WWW based time and date server

Copyright ©1999 DaveH Enterprises

- Listen on a well known TCP port.
- Accept a connection.
- Find out the current time and date
- Convert time and date to a string
- Send back some http headers (Content-Type)
- Send the string wrapped in HTML formatting.
- Close the connection.

**loop forever**

# Accessing our custom server.

- We can publish the URL to our server, or embed links to the server in other HTML documents.
- We need to make sure the server is always running (on the published host and port).
- Once we are famous we can include advertisements and make money!

# Another ~~Money Making Scheme~~ Example

- Keep track of how many times our server is hit each day.
- Report on the number of hits our server got on any day in the past!
- The reply now *does* depend on the request.
- We have to remember that the request comes from a HTTP client, so we need to accept HTTP requests.

# Time & Date Hit Server

- Each request comes as a string (URI) specifying a resource.

- Our requests will look like this:

`/mm/dd/yyyy`

**Y2K Compliant !**



- An example URL for our service:

`http://www.timedate.com:4567/01/17/1999`

- We will get a request like:

`GET /01/17/1999 HTTP/1.1`

# Fancy means \$\$\$

- We want to provide a table that lists the number of hits received each hour of the day in question

`timedate.com hit report for 01/17/1999`

<u>hour</u>	<u>number of hits</u>
12-1AM	4,320
1-2AM	18,986
2-3AM	246

# HTML Basics

- I assume everyone knows something about HTML.
  - If not: check the home page for some links.
- HTML Tables:
  - `<TABLE>` , `</TABLE>` start/end a table
  - `<TR>` , `</TR>` start/end a table row
  - `<TD>` , `</TD>` start/end a table cell
  - `<TH>` , `</TH>` start/end table header cell

# timedate.com Hit Table

```
<TABLE>
<TR>
  <TH>hour</TH>
  <TH>number of hits</TH>
</TR>
<TR>
  <TD>12-1AM</TD>
  <TD>4,320</TD>
</TR>
<TR>
  <TD>1-2AM</TD>
  <TD>18,986</TD>
</TR>
```

<u>hour</u>	<u>number of hits</u>
12-1AM	4,320
1-2AM	18,986
2-3AM	246

# New code

**loop forever**

- Listen on a well known TCP port.
- Accept a connection.
- Record the “hit” in database.
- Read request - parse request to month,day,year
- Lookup hits for month,day,year in database.
- Send back some http headers (Content-Type)
- Create HTML table and send back to client.
- Close the connection.
- Collect millions in advertising revenues.

# Drawbacks to Custom Server Approach

- We might have lots of ~~ideas~~ custom services.
  - Each requires dedicated address (port)
  - Each needs to include:
    - basic TCP server code
    - parsing HTTP requests
    - error handling
    - headers
    - access control (might want users to pay each time they check the time and date!)

## Another Approach

- Take a general purpose Web server (that can handle static documents) and have it *process* requested documents as it sends them to the client.
- The documents could contain commands that the server understands (the server includes some kind of interpreter).

# Example Smart Server

- Have the server read each HTML file as it sends it to the client.
- The server could look for this:  
`<SERVERCODE> some command </SERVERCODE>`
- The server doesn't send this part to the client, instead it interprets the command and sends the result to the client.
- Everything else is sent normally.

# Example Commands

```
<SERVERCODE> Time </SERVERCODE>
```

```
<SERVERCODE> Date </SERVERCODE>
```

```
<SERVERCODE> Hitlist </SERVERCODE>
```

```
<SERVERCODE> Include file </SERVERCODE>
```

```
<SERVERCODE> randomfile directory </SERVERCODE>
```

# Example Document

```
<TITLE>timedate.com Home Page</TITLE>
```

```
<H1 ALIGN=CENTER>Welcome to timedate.com</H1>
```

```
<SERVERCODE> include fancygraphic </SERVERCODE>
```

The current time is

```
<SERVERCODE> time </SERVERCODE>.<P>
```

Today is <SERVERCODE> date </SERVERCODE>.

Visit our sponser:

```
<SERVERCODE> random sponsor </SERVERCODE>
```

# Real Life - Server Side Includes

- Many real web servers support this idea (but not the syntax I've shown).
- Server Side Includes (SSI) provides a set of commands that a server will interpret.
- Typically the server is configured to look for commands only in specially marked documents (so normal documents aren't slowed down).

# SSI Directives

- SSI commands are called *directives*
- Directives are embedded in HTML comments. A comment looks like this:

```
<!-- this is an HTML comment -->
```

- A directive looks like this:

```
<!--#command parameter="arg"-->
```

# Some SSI Directives

**echo:** inserts the value of an environment variable into the page.

SSI servers keep a number of useful things in environment variables:

`DOCUMENT_NAME`, `DOCUMENT_URL`

Example usage:

This page is located at

```
<!--#echo var="DOCUMENT_URL" -->.
```

# SSI Directives

**include:** inserts the contents of a text file.

```
<!--#include file="banner.html">
```

**lastmod:** inserts the time and date that a file was last modified.

```
Last modified <!--#lastmod file="foo.html">
```

**exec:** runs an external program and inserts the output of the program.

```
Current users <!--#exec cmd="/usr/bin/who">
```

# SSI Examples

- There are a number of SSI examples you can play with on the course Web.
- If you want the server to handle SSI directives you should name the file `something.shtml` (not `something.html`)

# SSI Example - exec

```
<H1>Directory listing using the <CODE>#exec</CODE>  
  SSI directive</H1>
```

```
<PRE>
```

```
<!--#exec cmd="ls -al"-->
```

```
</PRE>
```

```
<HR>
```

```
<H3> Here I forgot to use the HTML  
  <CODE><PRE></CODE> tag:</H3>
```

```
<!--#exec cmd="ls -al"-->
```

# Another SSI Example

```
<!--#INCLUDE FILE="header"-->
```

It is now:

```
<!--#config timefmt="%I:%M %p (%Z)"-->
```

```
<!--#echo var="DATE_LOCAL"-->
```

```
<BR>
```

Today is:

```
<!--#config timefmt="%A, %B %e, %Y"-->
```

```
<!--#echo var="DATE_LOCAL"--><BR>
```

```
<!--#INCLUDE FILE="footer"-->
```

```
<!--#config timefmt="%D"-->
```

This file last modified

```
<!--#echo var="LAST_MODIFIED"-->
```

# More Power

- Some servers support elaborate scripting languages.
- Scripts are embedded in HTML documents, the server interprets the script:
  - Microsoft *Active Server Pages* (ASP)
    - JScript, VBScript, PerlScript
  - Netscape *LiveWire*
    - JavaScript, SQL connection library.
  - There are others...

# Server Mapping and APIs

- Some servers include a programming interface that allows us to extend the capabilities of the server by writing modules.
- Specific URLs are mapped to specific modules instead of to files.
- We could write our `timedate.com` server as a module and merge it with the web server.

# External Programs

- Another approach is to provide a standard interface between external programs and web servers.
  - We can run the same program from any web server.
  - The web server handles all the http, we focus on the special service only.
  - It doesn't matter what language we use to write the external program.

# Common Gateway Interface

- CGI is a standard interface to external programs supported by most (if not all) web servers.
- The interface that is defined by CGI includes:
  - Identification of the service (external program).
  - Mechanism for passing the request to the external program.

# CGI Programming

- We will focus on CGI programming.
- Project 4 will be a CGI program, you will create a web-based scheduling system.
- CGI programs are often written in scripting languages (perl, tcl), we will concentrate on C (although you can do project 4 in perl).