# Curly's Drive-Thru

## Serving Dynamic and Static Documents on the Web

This is a tour of Curly's Drive-Thru -- a sample demonstration of another stop on the so-called information superhighway, made particularly agonizing to read with the frequent use of fast-food metaphors.

## A Brief History of the Web

Using the term in its broadest sense, the Web refers to all the information available 'out there' on the net, whether it is being served by the old stand-bys such as gopher and ftp, or the dominant method of what is usually called the Web, HTTP (or hypertext transfer protocol for those who like the long version).

HTTP and HTML developed almost concurrently with gopher (and it is almost unfortunate that gopher gained the attention it did before HTTP and HTML caught on). Originally developed at CERN, a research laboratory in Europe, the Web did not really start to take off until a program called Mosaic was developed at the National Center for Supercomputing Applications at the University of Illinois in Urbana-Champaign. Mosaic was a graphical browser for the Web, and slowly the power of a graphical interface to the vast resources of the Internet was realized.

Since then, the Web has exploded, with HTTP traffic alone taking up progressively more of the bandwidth of the net, and basically drowning out everything, except possibly Usenet news. The commercial possibilities of the Web are just beginning to be realized, with smaller companies and technical companies leading the way, but with everyone else close on their heels (witness the presence of something as mundane as Zima on the Web).

## The Business Plan (HTTP and HTML)

As alluded to above, the primary type of document being pushed around the net as a Web document is called an HTML document. A derivative of SGML, HTML stands for Hypertext Markup Language, and it is a way of adding information to text about what sort of information each part is (a header, a list, or just part of a paragraph of text), some information about how to display it, hypertext links, and other information.

The primary method of transport for this information is called HTTP, or Hypertext Transfer Protocol. HTTP is a stateless protocol that is similar in many ways to the basic gopher protocol, but more flexible. Combined with the MIME standards, almost any sort of data can be transmitted using HTTP.

## Paving the Driveway (Installing a Server)

On Curly, I've elected to install the NCSA (National Center for Supercomputing Applications, located at the University of Illinois in Urbana-Champaign) Web server, called NCSA httpd. Three other highly-regarded servers include the CERN httpd, BSDI's plexus, and John Hopfield's WN. All four of these are excellent servers, but I selected to install the NCSA httpd because I have found it to be very cleanly coded (thus easy to change) and powerful enough for my needs without all sorts of extra features that I don't need (such as CERN httpd's support for acting as a proxy server, or WN's extensive indexing support).

NCSA httpd can be obtained from ftp.ncsa.uiuc.edu, and documentation is located on the Web from http://hoohoo.ncsa.uiuc.edu/docs/Overview.html. The most recent version as of this writing is 1.3R, which fixes the major security hole found in version 1.3, but version 1.4 is reportedly in the beta-testing stages and should be out "soon."

Because most of NCSA httpd's configuration is done through configuration files, and not at compilation time, compiling the server is very painless. One thing that must be decided before it is compiled, however, is where it is going to be installed. I decided to install Curly's Drive-Thru in /usr/local/www -- that's located on the partition with the most free space. This location is referred to as $HOME in the rest of this document.

## Wiring the Intercom (httpd.conf)

Most of the configuration for the server itself is done through the file $HOME/conf/httpd.conf. Things that are configured in this file include where various log files go, which port the server should serve documents from, which user the server should run as, and how the server is running (from inetd or standalone).

As I already mentioned, our Web service is based in /usr/local/www, so this is what our ServerRoot is set to in httpd.conf. Three main log files are generated by httpd: the transfer log ($HOME/logs/access), which logs each request sent to the server; the error log ($HOME/logs/errors), where the server logs an errors that it encounters; and the pid file, where the server records its process ID (which is useful for killing or restarting the server).

Our server runs standalone, as opposed to being started by inetd, because in a high-traffic environment, this is more efficient. If httpd were started by inetd instead of running standalone, it would have to parse its configuration files each time someone connected to the server, which causes an avoidable, thus undesirable, strain on the machine's CPU resources. Running the server standalone does inflict some small penalty in the form of memory being used by the server, but this is not a big deal because of the small memory footprint of NCSA's server.

Finally, although the server runs as root because it needs to connect to a priviliged port (80), it is smart enough to change personalities when it dives in the access files that it is going to serve. I have configured our server to turn itself into the "www" user and join the "www" group, which is the same user that owns all of the documents that are going to be served (convenient coincidince?).

## Arranging the Menu (srm.conf)

Another configuration file, srm.conf, controls certain aspects of how the server is going to serve documents. This is where you tell the server where the top of the document tree is (that is, the first file to serve to people when they access the site), where user's personal pages are stored, and various files and names to treat in special ways.

Here I've set it up so that the top of Curly's Drive-Thru is located in /usr/local/www/curly, and the document to serve when a user specifies only a directory and not a specific filename is index.html. This allows the use of shorter URL's such as http://curly.cs.hmc.edu/ instead of the more complete http://curly.cs.hmc.edu/index.html. I've also specified "www" as the directory in each user's home directory for their personal Web pages to be located. That is, if a user on Curly wants to offer a file on the Web called "mypage.html," they can place it in a www/ directory off their home directory and tell people to access it with a URL of the form http://curly.cs.hmc.edu/~user/mypage.html.

It's also possible to set up aliases for certain frequently-used directories, such as where a collection of icons are contained. Also, a command must be put in this file to tell the server where all of the "CGI" programs are located. These are explained more fully later, but basically they are programs that might generate HTML files on their own or process incoming data from users.

Lastly, most of the srm.conf file is dedicated to configuring how the server presents directories where no index.html file is located. What the NCSA httpd does is to build a menu of files from the directory, which pretty little pictures for files and directories.

## Setting the Prices (access.conf)

It is probably desirable for a server administrator to be able to regulate which users may access some files. For

example, we might want to limit some items on our server to only on-campus users and not just anyone from the rest of the Internet.

The access.conf file allows the server administrator to set up the default access permissions for directory hierarchies, and tell the server which of those permissions can be overridden by a file called .htaccess in subdirectories. Some of the things that can be allowed (or disallowed) include symbolic links, CGI scripts, server-side includes, and server-generated indexes. Usually one would want to be more strict with directories that the administrator might not have direct control over, especially user's directories.

Our server is set up fairly liberally, with symbolic links and indexes allowed anyplace, and with access allowed to everybody by default.

# Installing the Menu and Intercom (HTML and CGI)

As mentioned in the brief history of the Web given above, the primary type of document served over the Web is an HTML file. Such a file is very similar to an ordinary text file, but with special tags included to indicate where paragraphs start and stop, and where various hypertext links should be included. Because it's not really the focus of this document, and there are a plethora of other goods sources about writing HTML files, I won't go into any details about such things.

One thing worth mentioning, however, is the support provided by the NCSA httpd server for so-called "server-side" includes. This extends HTML files slightly by allowing them to contain "include" directives similar to what you might see in C source code. This is useful for doing things like including signatures on the bottom of each file that contain information about the author and when the document was last modified.

One of the most useful things that a Web server such as NCSA's httpd server can do is support the use of CGI programs. CGI stands for "Common Gateway Interface," and specifies how the server talks to auxiliary programs called 'gateways.' These programs can provide a window from the Web to other databases of information, such as HMC's UserInfo system. Typically, CGI programs have information passed to them by users through HTML forms, but they can also be used to do things like transparently serve the user different documents depending on where they are connecting from or what browser they are using.

To allow users the ability to have their own CGI applications hosted by Curly, I've installed a program called 'cgiwrap.' This program acts as a wrapper around user's CGI scripts, and does some very simple security/error checking to at least keep the user partially honest. It will not prevent the user form writing a CGI program that simply sends /etc/passwd, but there is also nothing to stop the user from simply emailing that same file, so there is really not much of an additional security hole exposed here. It does allow the user the chance to shoot himself or herself in the foot by allowing access to some of their own files unwittingly, so it's use should be monitored. Fortunately, cgiwrap creates a log of all the files that are accessed through it so user's CGI applications can be more thoroughly checked by hand to make sure they're not doing anything nasty. It's worth mentioning that NCSA httpd could (and maybe should) be modified to do the same sorts of checking, but it does not currently.

As an example of a CGI application, I've created a what I've called, rather simply, Jim's Quote Server,. It is a pair of CGI scripts that allow users to add quotes to an online collection of quotes, and access that database of quotes (currently by getting a random selection, but eventually to include searching the database). While not particularly fancy, the application is a good example of using perl to write CGI applications, and how to handle different aspects of the service through a small number of individual scripts (for example, the same script is used to generate the form for submissions and to handle submissions).

# Changing the Register Tape (Maintenance)

Once a Web server is up and running, the most time-consuming thing is adding documents and services. But, like

many of the other servers on everyone's system, httpd generates a slew of log messages that need to be handled in some fashion. In addition to the typical archiving and scanning-for-important-things that one does with log files, some people also choose to generate statistics about their Web server from their logs and make those statistics available on their server as well. There is an excellent package available called getstats which generates all sorts of tables of information, and probably even graphs, but I have not looked at installing it on Curly yet.

## Customer Satisfaction (A Note on Standards)

Because the technology of the Web is evolving so rapidly, it is interesting to watch how the standards have emerged. In the beginning, there were relatively people involved with HTTP, HTML, and other such issues of the Web, so very little was actually set in stone and most of what could be called standards were simply common practice. Many of the things that have since become standards, such as in-line images, were simply how the first person with the idea implemented them (in many cases the developers of NCSA Mosaic, one of the first major graphical Web browsers). Other standards such as CGI were developed once the need was recognized, and usually were simply developed by discussions between a few developers (in the case of CGI, the authors of NCSA httpd and the plexus server, mainly). Now that people have realized that there is money (and substantial amounts of it) to be made from the Web, there is much more politicking involved in developing standards. While NCSA Mosaic clearly violated existing standards (or at least standard practice) when it added support for inline images, not much of an outcry was raised because the standards weren't cemented and there were fewer people involved. When Netscape Communications (then Mosaic Communications) released the first version of their browser, Netscape, there was a rather large outcry about the features that it supported that were non-standard, such as the CENTER tag and the infamous BLINK. Netscape was blasted on the net for allegedly ignoring the HTML/2.0 and HTML3 standards under development.

This has continued with Netscape Communication's announcement and strong push for its security protocol called SSL. Many people on the net have been resistant to Netscape's proposal because they feel that Netscape is simply bullying the W3C (the World Wide Web Consortium, the organization responsible for the development of the Web) by confronting them with a protocol that is already implemented by the most popular browser, Netscape, which gives them a strategic advantage over their competitors. The discussions surrounding Netscape Communication's behavior in these issues is interesting because of the unique position of the company, and the untiring personal involvement of Marc Andressen, one of the original authors of NCSA Mosaic, and one of the founders of Netscape Communications.

Personally, I have adopted many of what have been dubbed "Netscapisms" in pages I've created because they should be simply ignored by most other browsers, and they are really nice for the vast number of users that use Netscape out there.

## Would you like fries with that? (Bibliography)

---