The Apache EBCDIC Port

Warning: This document has not been updated to take into account changes made in the 2.0 version of the Apache HTTP Server. Some of the information may still be relevant, but please use it with care.

Topics

Overview of the Apache EBCDIC Port

Version 1.3 of the Apache HTTP Server is the first version which includes a port to a (non-ASCII) mainframe machine which uses the EBCDIC character set as its native codeset.

(It is the SIEMENS family of mainframes running the BS2000/OSD operating system [1]. This mainframe OS nowadays features a SVR4-derived POSIX subsystem).

The port was started initially to

- prove the feasibility of porting the Apache HTTP server[2] to this platform
- find a "worthy and capable" successor for the venerable CERN-3.0[3] daemon (which was ported a couple of years ago), and to
- prove that Apache's preforking process model can on this platform easily outperform the accept-fork-serve model used by CERN by a factor of 5 or more.

This document serves as a rationale to describe some of the design decisions of the port to this machine.

Design Goals

One objective of the EBCDIC port was to maintain enough backwards compatibility with the (EBCDIC) CERN server to make the transition to the new server attractive and easy. This required the addition of a configurable method to define whether a HTML document was stored in ASCII (the only format accepted by the old server) or in EBCDIC (the native document format in the POSIX subsystem, and therefore the only realistic format in which the other POSIX tools like `grep` or `sed` could operate on the documents). The current solution to this is a "pseudo-MIME-format" which is intercepted and interpreted by the Apache server (see below). Future versions might solve the problem by defining an "ebcdic-handler" for all documents which must be converted.

Technical Solution

Since all Apache input and output is based upon the BUFF data type and its methods, the easiest solution was to add the conversion to the BUFF handling routines. The conversion must be settable at any time, so a BUFF flag was added which defines whether a BUFF object has currently enabled conversion or not. This flag is modified at several points in the HTTP protocol:

- **set** before a request is received (because the request and the request header lines are always in ASCII format)
- **set/unset** when the request body is received - depending on the content type of the request body (because the request body may contain ASCII text or a binary file)
- **set** before a reply header is sent (because the response header lines are always in ASCII format)
- **set/unset** when the response body is sent - depending on the content type of the response body (because the response body may contain text or a binary file)

Porting Notes

1. The relevant changes in the source are `#ifdef`'ed into two categories:

   `#ifdef CHARSET_EBCDIC`
   : Code which is needed for any EBCDIC based machine. This includes character translations, differences in contiguity of the two character sets, flags which indicate which part of the HTTP protocol has to be converted and which part doesn't *etc.*

   `#ifdef _OSD_POSIX`
   : Code which is needed for the SIEMENS BS2000/OSD mainframe platform only. This deals with include file differences and socket implementation topics which are only required on the BS2000/OSD platform.

2. The possibility to translate between ASCII and EBCDIC at the socket level (on BS2000 POSIX, there is a socket option which supports this) was intentionally *not* chosen, because the byte stream at the HTTP protocol level consists of a mixture of protocol related strings and non-protocol related raw file data. HTTP protocol strings are always encoded in ASCII (the `GET` request, any Header: lines, the chunking information *etc.*) whereas the file transfer parts (*i.e.*, GIF images, CGI output *etc.*) should usually be just "passed through" by the server. This separation between "protocol string" and "raw data" is reflected in the server code by functions like `bgets()` or `rvputs()` for strings, and functions like `bwrite()` for binary data. A global translation of everything would therefore be inadequate.

   (In the case of text files of course, provisions must be made so that EBCDIC documents are always served in ASCII)

3. This port therefore features a built-in protocol level conversion for the server-internal strings (which the compiler translated to EBCDIC strings) and thus for all server-generated documents. The hard coded ASCII escapes `\012` and `\015` which are ubiquitous in the server code are an exception: they are already the binary encoding of the ASCII `\n` and `\r` and must not be converted to ASCII a second time. This exception is only relevant for server-generated strings; and *external* EBCDIC documents are not expected to contain ASCII newline characters.

4. By examining the call hierarchy for the BUFF management routines, I added an "ebcdic/ascii conversion layer" which would be crossed on every puts/write/get/gets, and a conversion flag which allowed enabling/disabling the conversions on-the-fly. Usually, a document crosses this layer twice from its origin source (a file or CGI output) to its destination (the requesting client): `file -> Apache`, and `Apache -> client`.

   The server can now read the header lines of a CGI-script output in EBCDIC format, and then find out that the remainder of the script's output is in ASCII (like in the case of the output of a WWW Counter program: the document body contains a GIF image). All header processing is done in the native EBCDIC format; the server then determines, based on the type of document being served, whether the document body (except for the chunking information, of course) is in ASCII already or must be converted from EBCDIC.

5. For Text documents (MIME types text/plain, text/html *etc.*), an implicit translation to ASCII can

be used, or (if the users prefer to store some documents in raw ASCII form for faster serving, or because the files reside on a NFS-mounted directory tree) can be served without conversion.

Example:

to serve files with the suffix `.ahtml` as a raw ASCII `text/html` document without implicit conversion (and suffix `.ascii` as ASCII `text/plain`), use the directives:

```
AddType text/x-ascii-html .ahtml
AddType text/x-ascii-plain .ascii
```

Similarly, any `text/foo` MIME type can be served as "raw ASCII" by configuring a MIME type "`text/x-ascii-foo`" for it using `AddType`.

6. Non-text documents are always served "binary" without conversion. This seems to be the most sensible choice for, *.e.g.*, GIF/ZIP/AU file types. This of course requires the user to copy them to the mainframe host using the "`rcp -b`" binary switch.
7. Server parsed files are always assumed to be in native (*i.e.*, EBCDIC) format as used on the machine, and are converted after processing.
8. For CGI output, the CGI script determines whether a conversion is needed or not: by setting the appropriate Content-Type, text files can be converted, or GIF output can be passed through unmodified. An example for the latter case is the wwwcount program which we ported as well.

Document Storage Notes

Binary Files

All files with a `Content-Type:` which does not start with `text/` are regarded as *binary files* by the server and are not subject to any conversion. Examples for binary files are GIF images, gzip-compressed files and the like.

When exchanging binary files between the mainframe host and a Unix machine or Windows PC, be sure to use the ftp "binary" (`TYPE I`) command, or use the `rcp -b` command from the mainframe host (the `-b` switch is not supported in unix `rcp`'s).

Text Documents

The default assumption of the server is that Text Files (*i.e.*, all files whose `Content-Type:` starts with `text/`) are stored in the native character set of the host, EBCDIC.

Server Side Included Documents

SSI documents must currently be stored in EBCDIC only. No provision is made to convert it from ASCII before processing.

Apache Modules' Status

Module	Status	Notes
core	+	
mod_access	+	
mod_actions	+	
mod_alias	+	

Module	Status	Notes
mod_asis	+	
mod_auth	+	
mod_auth_anon	+	
mod_auth_dbm	?	with own libdb.a
mod_autoindex	+	
mod_cern_meta	?	
mod_cgi	+	
mod_digest	+	
mod_dir	+	
mod_so	-	no shared libs
mod_env	+	
mod_example	-	(test bed only)
mod_expires	+	
mod_headers	+	
mod_imap	+	
mod_include	+	
mod_info	+	
mod_log_agent	+	
mod_log_config	+	
mod_log_referer	+	
mod_mime	+	
mod_mime_magic	?	not ported yet
mod_negotiation	+	
mod_proxy	+	
mod_rewrite	+	untested
mod_setenvif	+	
mod_speling	+	
mod_status	+	
mod_unique_id	+	
mod_userdir	+	
mod_usertrack	?	untested

Third Party Modules' Status

Module	Status	Notes
mod_jserv[4]	-	JAVA still being ported.
mod_php3[5]	+	mod_php3 runs fine, with LDAP and GD and FreeType libraries.
mod_put[6]	?	untested
mod_session[7]	-	untested

URI References

[1] http://www.siemens.de/servers/bs2osd/osdbc_us.htm
[2] http://dev.apache.org/
[3] http://www.w3.org/Daemon/
[4] http://java.apache.org/
[5] http://www.php.net/
[6] http://hpwww.ec-lyon.fr/~vincent/apache/mod_put.html
[7] ftp://hachiman.vidya.com/pub/apache/