CS144: Content Encoding

MIME (Multi-purpose Internet Mail Extensions)

- Q: Only "bits" are transmitted over the Internet. How does a browser/application interpret the bits and display them correctly?

- MIME (Multi-purpose Internet Mail Extensions)
  - Standard ways to "transmit" multimedia content over the Internet
  - Originally developed for email attachments, but currently used for all Internet data transmission
- MIME types
  - Specified as "type/subtype". RFC2046 standard.
    - IANA (Internet Assigned Number Authority) manages the official registry of all media types
  - In HTTP, it is specified in "Content-Type" header
    - E.g., `Content-Type: text/html`
  - Popular types:
    - Text: text/plain, text/html, text/css, ...
    - Image: image/jpeg, image/png, image/gif, ...
    - Audio: audio/mpeg (.mp3), audio/mp4 (.mpa), ...
    - Video: video/mp4, video/H264, video/x-flv, ...
    - Application: application/pdf, application:/octet-stream, ...
    - Multipart: more on this later
- Q: What multimedia types/format should a browser support?
  - HTML5 is content-type/codec agnostic
  - No particular format support is required
  - Users expect browsers support "popular" codecs, such as JPG, PNG for images
  - Important legal issues regarding patent licensing
    - 1999 UNISYS patent claim on GIF (expired in 2003)
    - Licensing uncertainty for H.264 internet streaming
      - In 1997, MPEG/LA was formed to facilitate patent license after

MPEG2 standardization

- MPEG/LA's position on H.264 internet streaming: free through 2010!
- Google's purchase of VP8 patents (On2) in 2010

- H.265 (High Efficiency Video Coding): MPEG/LA vs HEVC Advance
  - Google's push for AV1 (AOMedia Video 1: Alliance for Open Media Video 1)

Text Encoding

- Q: How does a browser map a sequence bits to characters if it is text?

- Character encoding/Character set
  - Mapping between numeric numbers and alphabetic characters
  - Many different character encodings
- Early character encodings (until mid 90's)
  - ASCII (American Standard Code for Information Interchange)
    - created in 1963. First published as standard in 1967.
    - 7bits. defines codes for 128 characters
    - the basis of most of current encoding of roman characters
  - EBCDIC (Extended Binary Coded Decimal Interchange Code)
    - created in 1963 by IBM for IBM mainframes
    - 8bits. designed to be easy to represent in punch cards
    - still used by some IBM mainframes.
  - ISO-8859-1 (= Latin-1)
    - 8bits. consisting of 191 characters from the Latin script
    - ASCII non-control characters have the same encoding
    - used throughout Western Europe and America.
    - ISO-8859-15, Windows-1252: more characters for French, Estonian,...
  - Local/regional encoding
    - local character codes developed by each country
    - DBCS (Double Byte Code Character Set)
      - one or two bytes are used to represent a character
      - frequently used in Asia

        - Example: GB2312 (Simplified Chinese), EUC-KR (Korean), . . .

- Q: What are the problems of multiple encoding standards?

- *Code page* (= character encoding)
  - a unique number given to a particular character encoding by a system
    - On Windows: Hebrew (862), Greek (727), Korean (949)
  - OS sets the global code page for the computer
  - Q: What are the problems of a system-wide code-page setting?

UNICODE

- Motivation: Assign a unique number for every character in the world!
- International text encoding standard managed by Unicode Consortium
  - First standard, Unicode version 1.0 was published in October 1991
  - (almost) yearly release of a new Unicode version
- Every character maps to a CODE POINT
  - A -> U+0041
  - Hello -> U+0048 U+0065 U+006C U+006C U+006F.
- Originally defined to be a 16bit standard
  - No longer true. Currently 21bits (0x000000 – 0x10FFFFFF)
- a CODE POINT may be encoded into a sequence of bytes through an encoding scheme

UCS-2 (2-byte Universal Character Set)

- the first encoding scheme used for Unicode
- Represent the (original) unicode characters with two btes
  - U+0041 -> 00 41
- Unicode byte order mark: U+FEFF
  - little endian/big endian issue
  - gives hints on the endian mode
  - stored at the beginning of a Unicode string

- Used by many systems, including Windows, macOS, Java, .NET, . . .
- Q: Any problem with UCS-2 scheme?
  - space waste
  - Q: What will C program do for unicode-encoded data “a” (00 41)?
    - Legacy applications cannot handle UNICODE data even if data contains only ASCII characters
  - Q: What will a UNICODE program do for the input 41 42 43 44?
    - UNICODE applications cannot handle legacy ASCII data
  - Q: If one byte is lost in the middle from 00 41 00 42 00 43, how is it interpreted?
- UCS-2 did not take off much for internet applications

UTF-8

- Primary goal: backward compatibility with ASCII encoding
  - Both UTF-8 and ASCII encoding should map all ASCII characters to the *same* 1-byte number
    - e.g., A: U+0041 -> 41
    - Q: What is the benefit of this property?
  - Allow easy recovery of the string from error
    - even if a byte is missing, recover from the next character
- UTF-8 encoding standard

  0000 - 007F: 00000000 0zzzzzzz -> 0zzzzzzz
  0080 - 07FF: 00000yyy yyzzzzzz -> 110yyyyy 10zzzzzz
  0800 - FFFF: xxxxyyyy yyzzzzzz -> 1110xxxx 10yyyyyy 10zzzzzz

- Q: What will be UTF-8 encoding of character A (U+0041)?

- Example: 11010111 10111000 11101010 10111101 10110110 01111000
  - Q: How many characters in the example? How can we tell the beginning of a new character?
  - Q: How to recover if the second byte is lost during transmission?
- Q: If two strings are of the same length, are their encodings of the same length?
  - variable length encoding vs. fixed-length encoding
- UTF-8 encoding is the most popular encoding standard on Internet
  - Used by > 90% web sites

UTF-16

- Extension of UCS-2 after UNICODE became 21 bits
  - Two bytes are not enough to represent every unicode character!
  - Make the encoding as similar as possible to UCS-2, but allowing more characters
  - Variable length: either 2 bytes or 4 bytes
    - U+0000 to U+D7FF and U+E000 to U+FFFF: 2-byte encoding just like UCS-2
    - U+10000 to U+10FFFF: use 4-bytes to represent them
- Other Unicode encodings also exist
  - e.g., UTF-32: “32bit encoding”, . . .

Using UNICODE

Q: How can we use/specify UNICODE?

- HTTP: Text type character encoding is specified as the “charset” parameter
  - E.g., Content-Type: text/html; charset=UTF-8
- HTML5:

  - `<meta charset="utf-8">`
  - `&#0041;` for U+0041
  - For Web pages, UTF-8 encoding is by far the most popular encoding standard
- Most modern OS's support Unicode natively
  - Windows, macOS: UTF-16, Linux: UTF-8, . . .
- Most modern languages, like Java and Javascript, use unicode as the default string type
  - provide multiple encoding/decoding functions for UTF-8, UTF-16, ISO-8859-1,. . .
- But, unicode support in C++ is *messy*
  - On Linux, standard libraries, like std::string, support UTF-8
  - On Windows, UTF-16 can be used:
    - use character type `wchar_t` (wide char) instead of `char`
    - use wcs functions instead of str functions. (e.g., `wcslen` instead of `strlen`).
    - prefix string constant with L, like L"Hello"

References

- MIME: RFC 2046
  - IANA media type list: https://www.iana.org/assignments/media-types/media-types.xhtml
- Unicode standard: http://www.unicode.org/versions/latest/