Content Encoding

Junghoo Cho

cho@cs.ucla.edu

Content Encoding - Junghoo Cho - cho@cs.ucla.edu

Today's Topic

- MIME
- Text encoding standards
  - ASCII
  - UNICODE

Multimedia over Internet

- Q: Only “bits” are transmitted over the Internet. How does a browser/application interpret the bits and display them corr
- `Content-Type:` header
  - e.g., `Content-Type: text/html`

MIME (Multi-purpose Internet Mail Extensions)

- Standard way to indicate the type of the transmitted conten
  - text vs image vs video vs …
- Originally developed for email attachments, but currently us Internet data transmission
  - RFC2046
  - IANA (Internet Assigned Number Authority) manages the official re media types

MIME Type Specification

- Format: “type/subtype”
  - e.g., `text/html`
- Popular MIME types (case insensitive)
  - 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, …
  - Application: application/pdf, application/octet-stream, …
  - Multipart: more on this in a later lecture

Browser Support

- MIME type is specified in "Content-Type" HTTP header
  - E.g., `Content-Type: text/html`
- Q: What multimedia types/format should a browser support
- No particular support is required
  - HTML5 is content-type/codec agnostic
  - But users expect supports for "popular" codecs, such as JPG, PNG, et

Legal Issues

- 1999 UNISYS patent claim on GIF
- 2010 uncertainty on H.264 Web streaming
- Ongoing uncertainty on H.265
  - MPEG/LA vs HEVC Advance
  - Google's push for AV1

Text Encoding

- Q: For text, how does a browser map a sequence bits to char
- Character encoding/Character set
  - Numbers ↔ characters
  - Many character encoding standards exist

Early Standards

- ASCII (1963)
  - 7bits. 128 characters
  - extended to many 8-bit standards (e.g., ISO-8859-1)
  - basis of current standards for roman characters
- EBCDIC (1963)
  - create by IBM for IBM mainframes
  - 8bits. designed to be easy to represent in punch cards
  - still used by some IBM mainframes

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), …

Code Page

- Q: How does a computer know what encoding standard is us
  file in the system?
- Early solution: system-wide specification
  - OS sets the global code page for all files in the computer
- *Code page* (= character encoding)
  - a unique number given to a particular character encoding by a syste
  - On Windows: Hebrew (862), Greek (727)
- Q: Any problem with a system-wide code-page setting?

UNICODE

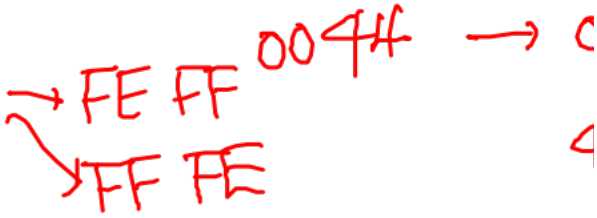
- Motivation:
  - One standard for all existing characters in the world
  - Assign a unique number for every character in the world!
- V1.0 was published in October 1991
  - managed by Unicode Consortium
  - (almost) yearly release of a new Unicode version

Code Point

- 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 – 0x10FFFFF)
- A CODE POINT is encoded into a sequence of bytes through encoding scheme

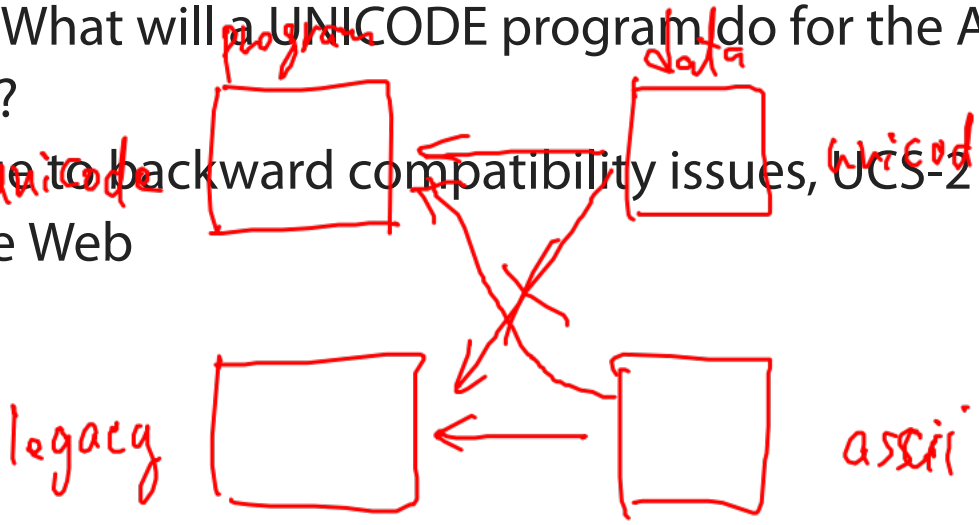
UCS-2 (2-byte Universal Character Set

- First Unicode encoding scheme
- Represent the (original) unicode characters with two bytes
  - 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



UCS-2 Problems

- Q: What will C program do for unicode-encoded data 'a' (00 4
- Q: What will a UNICODE program do for the ASCII text input 4
  44?
- Due to backward compatibility issues, UCS-2 did not take off
  the Web



UTF-8 to the Rescue

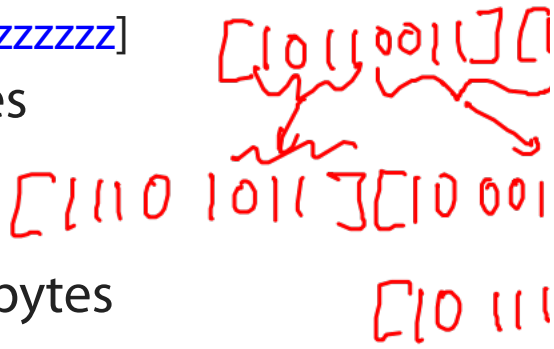
- We need to make Unicode backward compatible with ASCII!
- Q: But how?
- Idea:
  - Both UTF-8 and ASCII encoding should map all ASCII characters to t
    byte number
    - e.g., A: U+0041 → 41
    - Q: Why?

UTF-8: Variable-Length Encoding

- Use 1-4 bytes depending on the CODE POINT range
  - Variable length encoding

UTF-8 Encoding

- U+0000 - U+007F: encoded to 1 byte
  - [00000000] [0zzzzzzz] → [0zzzzzzz]
- U+0080 - U+07FF: encoded to 2 bytes
  - [00000yyy] [yyzzzzzz] → [110yyyyy] [10zzzzzz]
- U+0800 - U+FFFF: encoded to 3 bytes
  - [xxxxyyyy] [yyzzzzzz] → [1110xxxx] [10yyyyyy] [10zzzzzz]
- U+10000 - U+10FFFF: encoded to 4 bytes
  - [____wwwxx] [xxxxyyyy] [yyzzzzzz] → [11110www] [10xxxxxx] [10yyyyyy] [10zzzzzz]



UTF-8 Examples

- 'A': U+0041 (range 0000-007F)
  - [00000000] [01000001] → [01000001]
- 'Ɛ': U+0190 (range 0080-07FF)
  - [00000001] [10010000] → [11000110] [10010000]
- '한': U+D55C (range 0800-FFFF)
  - [11010101] [01011100] → [11101101] [10010101] [10011100]

UTF-8: Questions

- Q: How many bytes are used to represent an ASCII character
- All existing ASCII-encoded data is UTF-8 encoded!
  - Due to backward compatibility, UTF-8 is most popular on the Web
  - Used by > 90% web sites
- Q: If two texts have the same number of characters, do their encodings use the same number of bytes?
- UTF-8 is *variable-length* encoding

UTF-16

- Extension of UCS-2 to cover 21 bit code points
- Variable length: either 2 bytes or 4 bytes
  - U+0000 to U+D7FF and U+E000 to U+FFFF: 2-byte encoding just lik
  - U+10000 to U+10FFFF: 4-byte encoding
- Other Unicode encodings also exist
  - e.g., UTF-32: “32bit fixed-length encoding”, …

Using UNICODE (1)

- Q: How can we use UNICODE?
- HTTP
  - Character encoding is specified as the `charset` parameter of `Conten` header
  - E.g., `Content-Type: text/html; charset=UTF-8`
  - UTF-8 encoding is by far the most popular encoding standard
- HTML:
  - `&#x41;` for U+0041 (A)

Using UNICODE (2)

- Most modern OS's support Unicode natively
  - Windows, macOS: UTF-16, Linux: UTF-8, …
- Most modern languages, like Java, Javascript, and Python3, u
  unicode as the default string type
  - provide multiple encoding/decoding functions for UTF-8, UTF-16, IS

Using UNICODE (3)

- Unicode support in C++ is *messy*
  - On Unix, standard libraries, like std::string, support UTF-8
    - `wchar_t` means different things depending on the OS
  - Windows supports UTF-16:
    - `wchar_t` (wide char) instead of `char`
    - `wcs` functions instead of `str` functions.
      - e.g., `wcslen` instead of `strlen`
    - prefix string constant with L, like `L"Hello"`
  - Mac supports UTF-16
  - …

References

- MIME: RFC 2046
- IANA media type list
- Unicode