Lab Notes
Distributed Systems

External data representation

Marshalling and Unmarshalling

- The information stored in running programs is represented as data structures
  - e.g., by sets of interconnected objects

- The information in message consists of sequences of bytes.

- Irrespective of the form of communication used, the data structures must be
  - Flattened, converted to a sequence of bytes before transmission
  - Rebuilt on arrival

- *External data representation*
  - An agreed standard for the representation of data structures and primitive values

Sending data over the network

- The individual primitive data items transmitted in messages can be data values of many different types, and not all computers store primitive values such as integers in the same order.

- The representation of floating-point numbers also differs between architectures.

- Remote machine may have:
  - Different byte ordering
  - Different sizes of integers and other types
  - Different floating point representations
  - Different character sets
  - Alignment requirements

Marshalling and Unmarshalling

- *Marshalling* is the process of taking a collection of data items and assembling them into a form suitable for transmission in a message.
  - Marshalling consists of the translation of structured data items and primitive values into an external data representation.

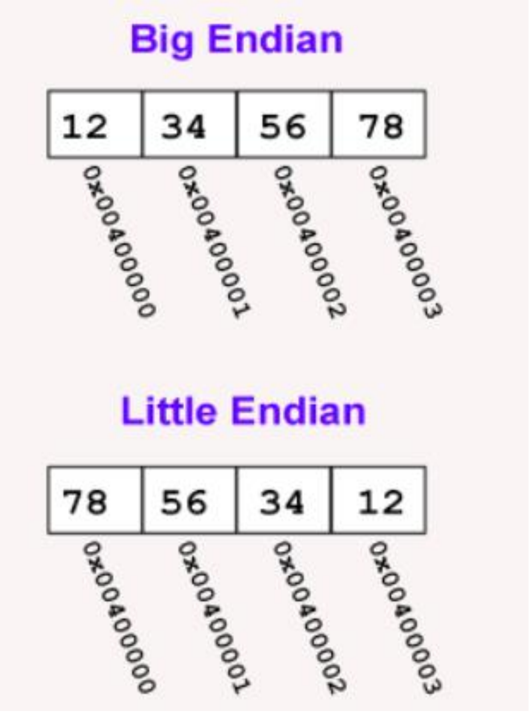
- *Unmarshalling* is the process of disassembling them on arrival to produce an equivalent collection of data items at the destination.
  - Unmarshalling consists of the generation of primitive values from their external data representation and the rebuilding of the data structures.

Marshalling and Unmarshalling: Integer ordering

- There are two variants for the ordering of integers:
  - **Big-endian** order
    - the most significant byte comes first; and
  - **Little-endian** order
    - the most significant byte comes last

- E.g., say that the 32-bit pattern 0x12345678 is stored at address 0x00400000.
  - the most significant byte is 0x12;
  - the least significant is 0x78.

- Some processors may operate in either mode
  - **Bi-endian**



Representing data

- IP (headers) use **big** endian byte ordering for 16- and 32-bit values

- Big endian
  - JVM, OpenRISC, Atmel AVR32, IBM z-series, SPARC < V9, older PowerPC, Motorola 680x0

- Little endian
  - Intel/AMD IA-32, x64

- Bi-endian
  - PowerPC, SPARC V9, MIPS, IA-64 (Intel Itanium), ARM

```
main() {
 unsigned int num;
 char *a = (char *)&num;

 num = 0x55667788;

 printf("%02x, %02x, %02x, %02x\n",
         a[0], a[1], a[2], a[3]);
}
```

- Output on IntelCPU:

```
88, 77, 66, 55
```

- Output on PowerPC

```
55, 66, 77, 88
```

Marshalling and Unmarshalling: Character Codes

- Another issue is the set of codes used to represent characters.

- For example
  - the majority of applications on systems such as UNIX use ASCII character coding, taking *one* byte per character, but
  - the Unicode standard allows for the representation of texts in many different languages and takes *two* bytes per character.

External data representation and marshalling

- The following methods can be used to enable any two computers to exchange binary data values:
  - The values are converted to an agreed external format before transmission and converted to the local form on receipt;
    - if the two computers are known to be the same type, the conversion to external format can be omitted.
  - The values are transmitted in the sender’s format, together with an indication of the format used, and the recipient converts the values if necessary.

Marshaling vs. serialization

- Marshaling uses serialization
- *Loosely* synonymous
- Serialization
  - converting an object data into a sequence of bytes that can be sent over a network
- Marshaling:
  - Converting *parameters* into a form that can be reconstructed (unmarshaled) by another process.
  - It may include object ID or other state.

External data representation: Approaches

- Java’s object serialization
  - which is concerned with the flattening and data representation of any single object or tree of objects that may need to be transmitted in a message or stored on a disk. It is for use only by Java.

- XML (Extensible Markup Language)
  - Defines a textual format for representing structured data.
  - It was originally intended for documents containing textual self-describing structured data
    - For example documents accessible on the Web
  - Now also used to represent the data sent in messages exchanged by clients and servers in web services.

XML definitions

- XML consists of tags and character data
- XML document is defined by pairs of tags enclosed in angle brackets.
- Person structure with value: {‘Smith’, ‘London’, 1984}

```
<person id="123456789">
           <name>Smith</name>
           <place>London</place>
           <year>1984</year>
           <!-- a comment -->
</person >
```

- <*name*> and <*place*> are both tags.
- As in HTML, layout can generally be used to improve readability.
- Comments are denoted in the same way as those in HTML.

XML: eXtensible Markup Language

Pros

- Human-readable
- Human-editable
- Interleaves structure with text (data)
- There are binding libraries for lots of languages.
- A good choice if you want to share data with other applications/projects

Cons

- Verbose
  - Transmit more data than needed
  - Space intensive
- Data conversion always required for numbers
- Encoding/decoding
  - Can impose a huge performance addition on applications.
  - Longer parsing time
- Navigating an XML DOM tree is considerably more complicated than navigating simple fields in a class

External data representation: other techniques

- Protocol buffers
  - Google uses an approach called *protocol buffers (aka* protobuf) to capture representations of both stored and transmitted data
  - offers a common serialization format for Google, including the serialization of requests and replies in remote invocation

- JSON (JavaScript Object Notation)
  - an approach to external data representation [www.json.org].

- Protocol buffers and JSON
  - more lightweight approaches to data representation
    - when compared, for example, to XML.

JSON (JavaScript Object Notation)

- Lightweight (relatively efficient) data interchange format
  - Lighter alternative to XML
- Based on JavaScript
- Human writeable and readable
- Self-describing (explicitly typed)
- Language independent
- Easy to parse

JSON

- Derived from JavaScript that is used in web services and other connected applications.
- Browsers can parse JSON into JavaScript objects natively.
- On the server, JSON needs to be parsed and generated using JSON APIs.

Uses of JSON

- Ajax applications
- Configurations
- Databases
- RESTful web services:
  - All popular websites offer JSON as the data exchange format with their RESTful web services.
  - RESTful web services are web services which are REST based.
    - Representational State Transfer (REST) is an approach in which clients use URLs and the HTTP operations GET, PUT, DELETE and POST to manipulate resources that are represented in XML.
    - The emphasis is on the manipulation of data resources rather than on interfaces.

Protocol Buffers (*protobuf*)

- A mechanism for serializing structured data
- Similar to XML
  - smaller, faster, and simpler
- Uses binary format
  - rather than text format of XML and JSON
- Is in fact an IDL (Interface Definition Language)

Google Protocol Buffers

- Properties:
  - Efficient, binary serialization
  - Support protocol evolution
    - Can add new parameters
    - Order in which parameters are specified is not important
    - Skip non-essential parameters
  - Supports types, which give you compile-time errors
  - Supports quite complex structures

- Usage:
  - It is a binary encoding format that allows you to specify a *schema* for your data
  - Protocol buffers are used for other things, e.g., serializing data to non-relational databases – their backward-compatible feature make them suitable for long-term storage formats

- As well as being language- and platform-neutral, protocol buffers are also agnostic with respect to the underlying RPC protocol - compatible with many types.

Google Protocol Buffers vs XML

- Simpler format compared to XML, faster in operation

- But, Google infrastructure is a relatively closed system
  - It does not address interoperability across open systems
    - XML does

- XML is significantly richer
  - it generates self-describing messages that contain the data and associated metadata describing the structure of the messages
  - Protocol buffers do not provide this facility directly

References

- Chapter 4: Coulouris, Dollimore and Kindberg,
  Distributed Systems: Concepts and Design

- http://docs.oracle.com/javase/8/docs/platform/serialization/spec/serialTOC.html
- https://people.cs.rutgers.edu/~pxk/417/notes/pdf/02b-encoding-slides.pdf
- https://chortle.ccsu.edu/AssemblyTutorial/Chapter-15/ass15_3.html