ECE 3574: Applied Software Design

Message Serialization

Today we are going to see various techniques for serializing objects, converting them to/from byte streams.

The ability to exchange objects over files, pipes, sockets, and shared memory is a common task in concurrent programming.

- Serialization (Marshalling or Pickling)
- Deserialization (Unmarshalling or UnPickling)
- QDataStream
- Google Protocol buffers

In general how do we make an object in memory persistent? We write it to a file.

A file is a linear stream of bytes that may be

- text-based: human-readable/editable, very portable and future proof, but we have to write parsers, they can be inefficient, can loose precision, take care to handle utf-8
- binary: machine-readable only, have to take care to make portable (Endianess), can easily get lost over time, but can preserve precision

To recover the object we read the file back in.

Example: ASCII text-based file

Suppose we want to store some information about a task in some Todo app.

```
struct Task
{
  std::string description;
  bool completed;
  char priority;
  std::string context;
};
```

Let's store this as an ASCII text file.
See example code: `task_text_delimited.cpp`

There are a few standard text file formats

- delimited files: common delimiters ',', ':', '|'
- INI files (originally from windows)
- XML (heavy, usually overkill)
- JSON (similar to xml but lighter)

JSON can handle UTF encoding, arrays, many data types, and there are light-weight open-source parsing libraries for it (I like `jsoncpp`).

Example: binary formatted file

Binary files use sequences of bytes to store the same representation as in memory.
Open the file in binary mode, read/write using read and write methods.
See example code: `write_data.cpp` and `read_data.cpp`.
Note:

- files written this way are often not portable across platforms (without additional formatting work).
- often you write a "magic number" at the beginning to tag the file as one you can read.

This approach is OK if you don't care about sharing the files, e.g. for message passing on the same machine or saving intermediate program state (undo or temporary backups).

Serializing pointer-based objects (linked-lists, pointer-based trees, etc)

Serializing non-linear data structures requires establishing a linear ordering.

Example Writers:

- Linked-List -> array -> write
- Binary Search Tree (layout discarded) -> pre-order traversal -> array -> write
- Binary Trees (layout preserved) -> Complete Binary Tree -> pre-order traversal -> array -> write

Example Readers:

- read -> array -> Linked-List
- read -> array -> inserts -> Binary Search Tree (layout discarded)
- read -> array -> pre-order traversal -> Complete Binary Tree -> Binary Trees (layout preserved)

Unless absolutely necessary do not create your own binary format.

Use a standard one.
Examples:

- Raster Images: png, jpeg
- Vector Images: svg, ps/pdf
- general 2D/3D shapes: stl, vrml
- General data: Hierarchical Data Format (HDF)

Summary

- Prefer text-based formats to binary, binary formats are fragile
- In either case, *use an existing standard if at all possible*

"Custom" file formats are evil!

From files to messages

Now that we can serialize and deserialize objects to/from files, we use the same mechanism for message passing.
To send an in-memory object:

- first, serialize it
- then, send it

To receive an in-memory object:

- receive it
- then, unserialize it

See example using unix pipe.

Serialization using QDataStream

`QDataStream` is a Qt class that can serialize/de-serialize many Qt objects into a platform independent binary stream.
These can be written to or read from any `QIODevice`, e.g. files or sockets.
Perfect solution to message passing (within Qt apps).
See example code.

There are many other serialization libraries

Examples:

- Google Protocol Buffers: “language-neutral, platform-neutral extensible mechanism for serializing structured data.” Uses a code generator.
- Cap’n Proto: fast, but requires a library specific memory layout
- Apache Thrift
- eProsima Fast Buffers

Next Actions and Reminders

- Read about C++11 Threading