22IT402 – DATA STRUCTURES II

UNIT V – LP20 – CIRCULAR BUFFER, MARSHALLING/UNMARSHALLING & JSON

1. What is a ring buffer?

The ring buffer (also known as a circular buffer, circular queue, or cyclic buffer) is a circular software queue. This queue has a first-in-first-out (FIFO) data characteristic. These buffers are quite common and are found in many embedded systems. Usually, most developers write these constructs from scratch on an as-needed basis.

The C++ Language has the Standard Template Library (STL), which has a very easy-to-use set of class templates. This library enables the developer to create the queue and other lists relatively easily. For the purposes of this article, however, I am assuming that we do not have access to the C++ language.

The ring buffer usually has two indices to the elements within the buffer. The distance between the indices can range from zero (0) to the total number of elements within the buffer. The use of the dual indices means the queue length can shrink to zero, (empty), to the total number of elements, (full). **Figure 1** shows the ring structure of the ring buffer, (FIFO) queue.

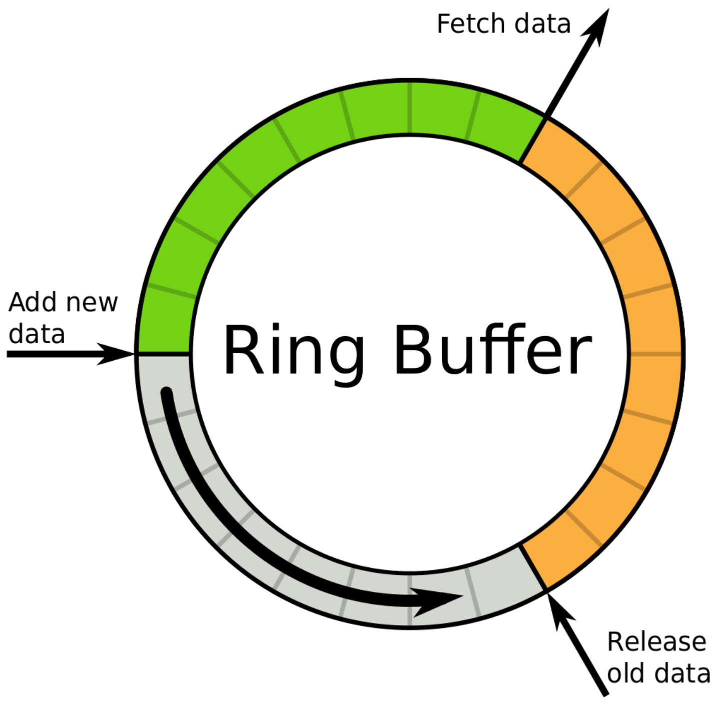


Figure 1: Structure of a ring buffer.

The data gets PUT at the head index, and the data is read from the tail index. In essence, the newest data “grows” from the head index. The oldest data gets retrieved from the tail index. **Figure 2** shows how the head and tail index varies in time using a linear array of elements for the buffer.

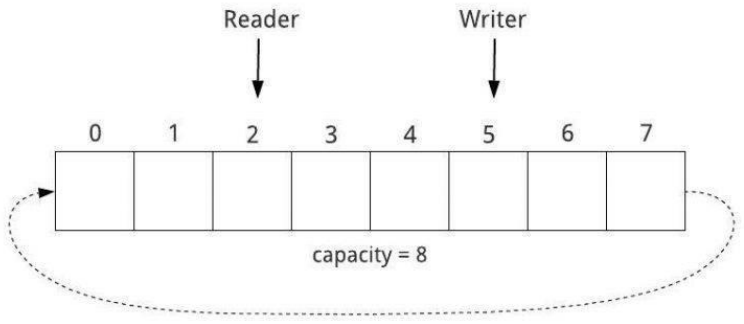


Figure 2: Linear buffer implementation of the ring buffer.

What is the ring buffer used for?

Single process to single process

In general, the queue is used to serialize data from one process to another process. The serialization allows some elasticity in time between the processes. In many cases, the queue is used as a data buffer in some hardware interrupt service routine. This buffer will collect the data so that at some later time another process can fetch the data for further processing. This use case is the single process to process buffering case.

This use case is typically found as an interface between some very high priority hardware service buffering data to some lower priority service running in some background loop. This simple buffering use case is shown in **Figure 3**.

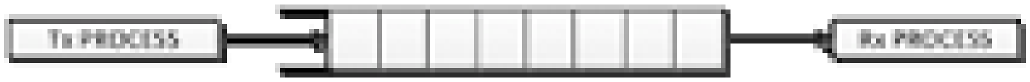


Figure 3: A single process to process buffer use case

In many cases, there will be a need for two queues for a single interrupt service. Using multiple queues is quite common for device drivers for serial devices such as RS-232, I2C or USB drivers.

Multiple processes to single process

A little less common is the requirement to serialize many data streams into one receiving streams. These use cases are quite common in multi-threaded operating systems. In this case, there are many client threads requesting some type of serialization from some server or broker thread. The requests or messages are serialized into a single queue which is received by a single process. Figure 4 shows this use case.

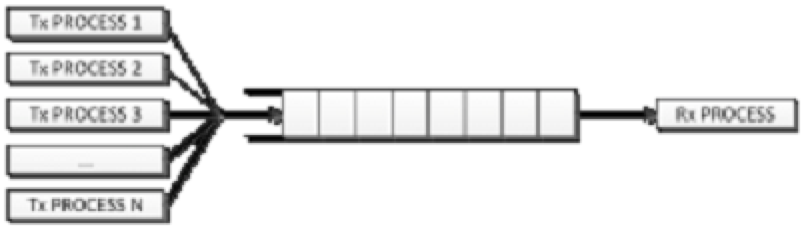


Figure 4: Multiple processes to process use case.

Single process to multiple processes

The least common use case is the single process to multiple processes case. The difficulty here is to determine where to steer the output in real time. Usually, this is done by tagging the data elements in such a way that a broker can steer the data in some meaningful way. **Figure 5** shows the single process to multiple processes use case. Since queues can be readily created, it is usually better to create multiple queues to solve this use case than it would be to use a single queue.

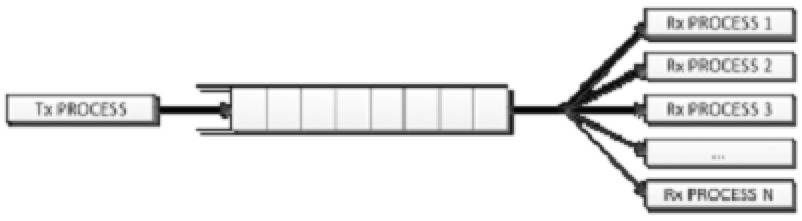


Figure 5: Single process to multiple processes use case.

Figure 6 shows how to reorganize the single process to multiple process use case using a set of cascaded queues. In this case, we have inserted an Rx / Tx Broker Dispatcher service, which will parse the incoming requests to each of the individual process queues.

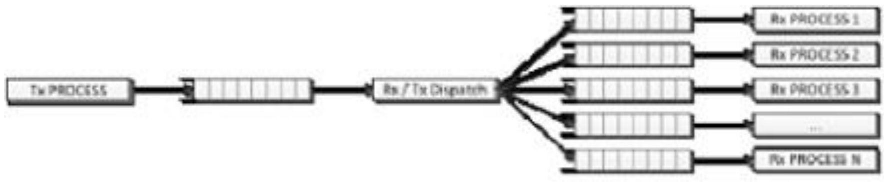


Figure 6: Single process to multiple process use case using a dispatcher and multiple queues.

Managing overflow

One must be able to handle the case where the queue is full and there is still incoming data. This case is known as the overflow condition. There are two methods which handle this case. They are to drop the latest data or to overwrite the oldest data. Either style may be implemented. In our case, I will use the drop latest incoming data method.

Applications

Ring Buffers are common data structures frequently used when the input and output to a data stream occur at different rates.

- Buffering Data Streams
- Computer Controlled Trafficking signal systems
- Memory Management
- CPU scheduling

2.Overview of Marshalling & Un-marshalling

Data structures are used to represent the information held in running applications. The information consists of a sequence of bytes in messages that are moving between components in a distributed system. So, conversion is required from the data structure to a sequence of bytes before the transmission of data. On the arrival of the message, data should also be able to be converted back into its original data structure.

Different types of data are handled in computers, and these types are not the same in every position where data must be transmitted. Individual primitive data items can have a variety of data values, and not all computers store primitive values like integers in the same order. Different architectures also represent floating-point numbers differently. Integers are ordered in two ways, big-endian order, in which the Most Significant Byte (MSB) is placed first, and little-endian order, in which the Most Significant Byte (MSB) is placed last or the Least Significant Byte (LSB) is placed first. Furthermore, one more issue is the set of codes used to represent characters. Most applications on UNIX systems use ASCII character coding, which uses one byte per character, whereas the Unicode standard uses two bytes per character and allows for the representation of texts in many different languages.

There should be a means to convert all of this data to a standard format so that it can be sent successfully between computers. If the two computers are known to be of the same type, the external format conversion can be skipped otherwise before transmission, the values are converted to an agreed-upon external format, which is then converted to the local format on receiving. For that, values are sent in the sender's format, along with a description of the format, and the recipient converts them if necessary. It's worth noting, though, that bytes are never changed during transmission. Any data type that can be supplied as a parameter or returned, as a result, must be able to be converted and the individual primitive data values expressed in an accepted format to support Remote Procedure Call (RPC) or Remote Method Invocation (RMI) mechanisms. So, an external data representation is a standard for representing data structures and primitive values that have been agreed upon.

- **Marshalling:** Marshalling is the process of transferring and formatting a collection of data structures into an external data representation type appropriate for transmission in a message.
- **Unmarshalling:** The converse of this process is unmarshalling, which involves reformatting the transferred data upon arrival to recreate the original data structures at the destination.

3.JSON (JavaScript Object Notation)

JSON stands for **J**ava**S**cript **O**bject **N**otation. It is a format for structuring data. This format is used by different web applications to communicate with each other. JSON is the replacement of the XML data exchange format in JSON. It is easy to struct the data compare to XML. It supports data structures like arrays and objects and the JSON documents that are rapidly executed on the server. It is also a Language-Independent format that is derived from JavaScript. The official media type for the JSON is application/json and to save those file **.json** extension.

Features of JSON:

- **Easy to understand:** JSON is easy to read and write.
- **Format:** It is a text-based interchange format. It can store any kind of data in an array of video, audio, and image anything that you required.
- **Support:** It is light-weighted and supported by almost every language and OS. It has a wide range of support for the browsers approx each browser supported by JSON.
- **Dependency:** It is an Independent language that is text-based. It is much faster compared to other text-based structured data.

JSON Syntax Rules: Data is in name/value pairs and they are separated by commas. It uses curly brackets to hold the objects and square brackets to hold the arrays.

Example:

Javascript

```
{    "
Courses": [
      {
         "Name" : "Java Foundation",
         "Created by" : "Geeksforgeeks",
         "Content" : [ "Java Core", "JSP",
                  "Servlets", "Collections" ]
      },
      {
         "Name" : "Data Structures",
         "also known as" : "Interview Preparation Course",
         "Topics" : [ "Trees", "Graphs", "Maps" ]
      }
   ]
}
```

Advantages of JSON:

- JSON stores all the data in an array so data transfer makes easier. That’s why JSON is the best for sharing data of any size even audio, video, etc.
- Its syntax is very easy to use. Its syntax is very small and light-weighted that’s the reason that it executes and response in a faster way.
- JSON has a wide range for the browser support compatibility with the operating systems, it doesn’t require much effort to make it all browser compatible.
- On the server-side parsing the most important part that developers want, if the parsing will be fast on the server side then the user can get the fast response, so in this case JSON server-side parsing is the strong point compare tot others.

Disadvantages of JSON:

- The main disadvantage for JSON is that there is no error handling in JSON, if there was a slight mistake in the JSON script then you will not get the structured data.

- JSON becomes quite dangerous when you used it with some unauthorized browsers. Like JSON service return a JSON file wrapped in a function call that has to be executed by the browsers if the browsers are unauthorized then your data can be hacked.
- JSON has limited supported tools that we can use during JSON development.

JSON stands for JavaScript Object Notation which is a lightweight text-based open standard designed which is easy for human-readable data interchange. In general, JSON is extended from JavaScript. JSON is language-independent and It is easy to read and write. The file extension of JSON is **.json**.

Example – JSON format

- In the below given example, you will see how you can store values in JSON format. Consider student information where Stu_id, Stu_Name, Course is an entities you need to store then in **JSON format** you can store these values in key values pair form. Let's have a look.

```
{
  "Student": [

    {
  "Stu_id"   :  "1001",
  "Stu_Name" :  "Ashish",
  "Course"   :  "Java",
    },

    {
      "Stu_id"   :  "1002",
      "Stu_Name" :  "Rana",
      "Course"   :  "Advance Java",
    }
   ]
  }
```

It is the method by which we can access means read or write JSON data in Java Programming Language. Here we simply use the **json.simple** library to access this feature through Java means we can encode or decode JSON Object using this **json.simple** library in Java Programming Language. Now, the json.simple package for Java contains the following files in it. So to access we first have to install json.simple package.

Step 1: Download the **json.simple** using the link:

Step 2: There is one more method to add the Maven dependency, so for that, we have to add the code given below to our *pom.xml* file.

```
<dependency>
  <groupId>com.googlecode.json-simple</groupId>
  <artifactId>json-simple</artifactId>
  <version>1.1</version>
</dependency>
```

The above-downloaded **.jar** file contains these Java source files in it:

```
// .jar file
META-INF/MANIFEST.MF
```

- org.json.simple.ItemList.class
- org.json.simple.JSONArray.class
- org.json.simple.JSONAware.class
- org.json.simple.JSONObject.class
- org.json.simple.JSONStreamAware.class
- org.json.simple.JSONValue.class
- org.json.simple.parser.ContainerFactory.class
- org.json.simple.parser.ContentHandler.class
- org.json.simple.parser.JSONParser.class
- org.json.simple.parser.ParseException.class
- org.json.simple.parser.Yylex.class
- org.json.simple.parser.Yytoken.class

JSON Object Encoding in Java: As we discussed above, this **json.simple** library is used to read/write or encode/decode JSON objects in Java. So let's see how we can code for encoding part of the JSON object using ***JSONObject* function**. Now we create a java file **mainEncoding.java** and save the below-written code in it.

```
import org.json.simple.JSONObject;
 // Program for print data in JSON format.
public class JavaJsonEncoding {
    public static void main(String args[])
    {
        // In java JSONObject is used to create JSON object
        // which is a subclass of java.util.HashMap.

        JSONObject file = new JSONObject();
        file.put("Full Name", "Ritu Sharma");
        file.put("Roll No.", new Integer(1704310046));
        file.put("Tuition Fees", new Double(65400));

        // To print in JSON format.
        System.out.print(file);
    }
}
```

Output:

- {"Full Name":"Ritu Sharma", "Roll No.":1704310046, "Tuition Fees":65400}

Now we will see how we can code for decoding part of the JSON object using ***JSONObject*function**. Now we create a java file **mainDecoding.java** and save the below-written code in it.

```
import org.json.simple.JSONObject;
import org.json.simple.JSONValue;

public class JavaJsonDecoding {

    public static void main(String[] args)
    {
        // Converting JSON data into Java String format
        String k = "{\"Full Name\":\"Ritu Sharma\",
       \"Tuition Fees\":65400.0, \"Roll No.\":1704310046}";
        Object file = JSONValue.parse(k);

        // In java JSONObject is used to create JSON object
        JSONObject jsonObjectdecode = (JSONObject)file;

        // Converting into Java Data type
        // format From Json is the step of Decoding.
        String name
            = (String)jsonObjectdecode.get("Full Name");
        double fees
            = (Double)jsonObjectdecode.get("Tuition Fees");
        long rollno
            = (Long)jsonObjectdecode.get("Roll No.");
        System.out.println(name + " " + fees + " "
                           + rollno);
    }
}
```

Output:

- Ritu Sharma 65400.0 1704310046

4.Tag: json

The wcf:json tag converts a specified Java object to JavaScript Object Notation (JSON).

Tag information

Tag information	
Body Content	empty

Attributes

Attributes

Attribute	Required	Request-time	Type	Description
object	true	true	java.lang.Object	The Java object that is to be conv

If the specified object is a Java boolean, number, string, date, character, class, or throwable class object, the tag json converts the object to a string by calling the toString() method of the tag. The resulting object is URL encoded. If the object is an SDO data object, the object is parsed and a structured JSON document is created.

Variables

No variables are defined for the wcf:json tag.

Limitations

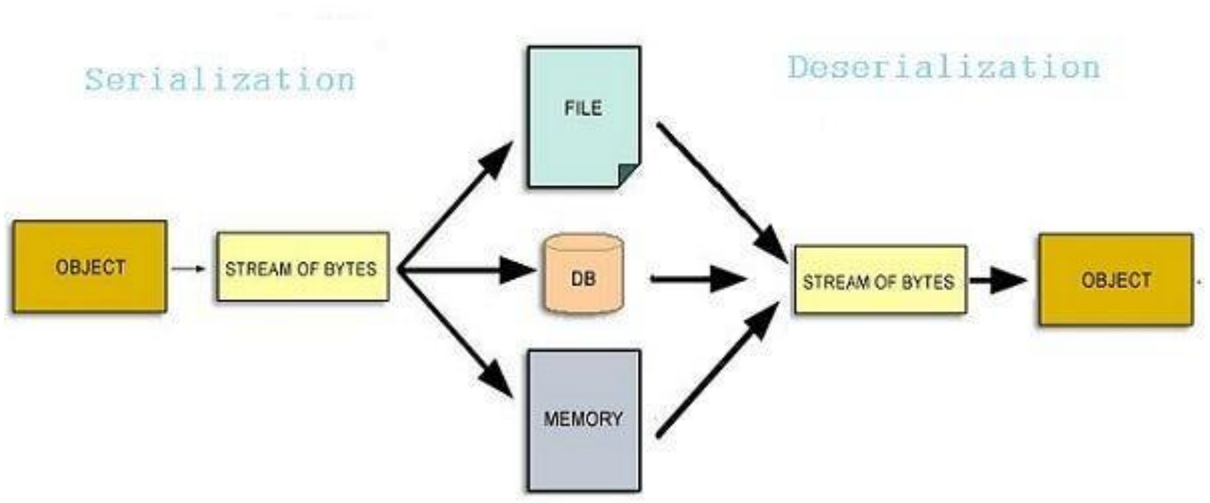
1. The wcf:json tag operates on Java objects, not Java primitives.
2. It cannot convert objects that reference themselves. That is, if an object references itself, a stack overflow error occurs.

Example

The following example converts the RequestProperties object into JSON notation:

```
<wcf:json object="${RequestProperties}"/>
```

5.JSON vs Protocol Buffer Simplified



Data serialization is the process of converting structured data to a format **that allows sharing or storage of the data** in a form that allows recovery of its original structure. Data De-serialization is the exact opposite. Various Data Serialization formats include XML, CSV, YAML, JSON, Protobuf etc.

- **Protocol Buffers** usually referred to as Protobuf, was internally developed by Google with the goal to provide a better way, compared to XML, for data serialization -deserialization. So they focused on making it simpler, smaller, faster and more maintainable then XML. But, this protocol even surpassed JSON with better performance, better maintainability, and smaller size.

- **JSON** (JavaScript Object Notation) is a lightweight data-interchange format and is based on a subset of the JavaScript Programming Language. **JSON** is “self-describing” and easy to understand.

Major Differences:

- JSON is a **text data** format independent of the platform.

```
{
   "quiz": {
      "sport": {
         "q1": {
            "question": "Which is correct team name in NBA?",
            "options": [
               "Golden State Warriros",
               "Huston Rocket"
            ],
            "answer": "Huston Rocket"
         }
      },
      "maths": {
         "q1": {
            "question": "5 + 7 = ?",
            "options": [
               "10",
               "11",
               "12",
               "13"
            ],
            "answer": "12"
         },
         "q2": {
            "question": "12 - 8 = ?",
            "options": [
               "1",
               "2",
               "3",
               "4"
            ],
            "answer": "4"
         }
      }
   }
}
//File source: https://support.oneskyapp.com/hc/en-us/articles/208047697-JSON-sample-files
```

- Protobuf uses **binary message format** that allows programmers to specify a schema for the data. It also includes a set of rules and tools to define and exchange these messages. A schema for a particular use of protocol buffers associates data types with field names, using integers to identify each field.

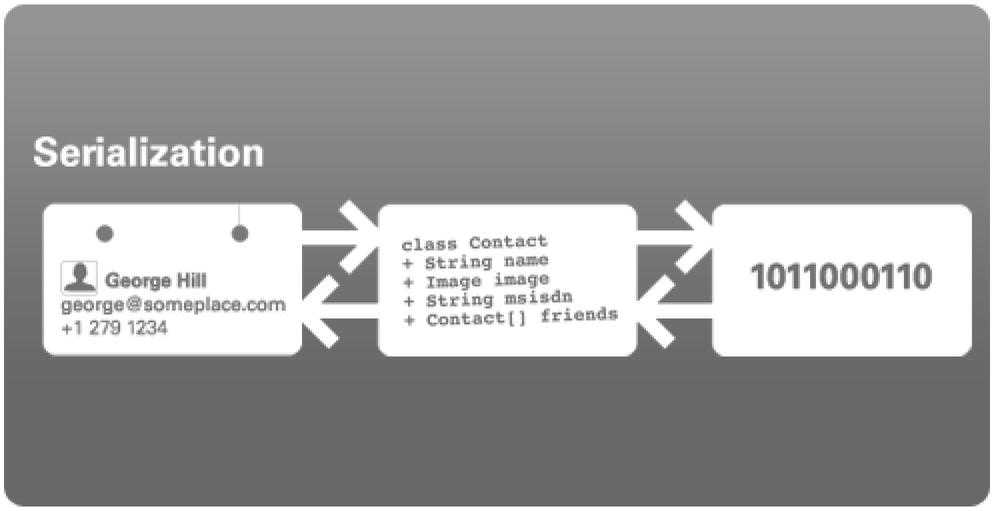
```
//A simple Proto file - Polyline.proto
syntax = “proto2”;
message Point {
 required int32 x = 1;
 required int32 y = 2;
 optional string label = 3;
}message Line {
 required Point start = 1;
 required Point end = 2;
 optional string label = 3;
}message Polyline {
 repeated Point point = 1;
 optional string label = 2;
}
//File source: https://en.wikipedia.org/wiki/Protocol_Buffers
```

- As **JSON** is textual, its integers and floats can be slow to encode and decode. **JSON** is not designed for numbers. Also, Comparing strings in **JSON** can be slow.

- **Protobuf** is easier to bind to objects and faster.

- **JSON** is widely accepted by almost all programming languages and highly popular.

- **Protocol buffers** currently support generated code in Java, Python, Objective-C, and C++. With proto3 language version, one can also work with Dart, Go, Ruby, and C#, with more languages to come.



Protobuf binary format Serialization

Advantages of Protobuf:

- **Simpler, faster, smaller** in size.

- **RPC support:** Server RPC interfaces can be declared as part of protocol files.

- **Structure validation:** Having a predefined and larger structure, when compared to JSON, set of data types, messages serialized on Protobuf can be automatically validated by the code that is responsible to exchange them.

Why use JSON aka disadvantages of Protobuf?

- **Non-human readability:** JSON, as exchanged on text format and with simple structure, is easy to be read and analyzed by humans. This is not the case with a binary format. [There are now ways to make protobuf human readable too though.]

- **Lesser resources and support:** You won’t find that many resources (do not expect very detailed documentation, nor too many blog posts) about using and developing with Protobuf.

- **Smaller community:** Probably the root cause of the first disadvantage. On Stack Overflow, for example, you will find roughly 1.500 questions marked with Protobuf tags. While JSON has more than 180 thousand questions on this same platform.