Sarojini Nama

Questions asked on GStreamer Training presentation

1. In Design Principles of GStreamer, what does extensible stands for?

Ans: Extensible is one of the Design Princple of the GStreamer. It means, all the plugins which are loaded dynamically can be extended and upgraded independently. And also, all the GStreamer objects can be extended using the GObject inheritance methods.

2. What are the advantages of GStreamer?

Ans: We all know GStreamer is used mostly for Application development, as it is very strong in coding development. Apart from this, other advantages of GStreamer are:

- Cross-platform Support
- Extensibility
- Wide range of plugins
- Support for various multimedia formats
- Integration with other communities
- Flexible pipeline architecture and
- Hardware acceleration support

3. Is there any alternative for GStreamer?

Ans: Yes, **FFmpeg** is one of the alternative for GStreamer. FFmpeg is primarily used through Command Line Interface. It is same as GStreamer which is used for multimedia development.

4. Difference between caps and properties.

Ans: Caps: Caps is nothing but the short name for "capabilities". These capabilities are attached to the pad templates and to pads. For pad templates, it will describe the types of media that may stream over a pad created from this template. For pads, it can either be a list of possible caps (usually a copy of the pad template's capabilities), in which case the pad is not yet negotiated, or it is the type of media that currently streams over this pad, in which case the pad has been negotiated already.

```
Pad Templates:
  SINK template: 'sink'
    Availability: Always
    Capabilities:
      audio/x-raw
                format: { (string)F64LE, (string)F64BE, (string)F32LE, (string)F32BE, (string)S32LE, (string)S32BE, (string)U32LE, (string)U32BE, (st
ring)S24_32LE, (string)S24_32BE, (string)U24_32LE, (string)U24_32BE, (string)S24LE, (string)S24BE, (string)U24LE, (string)U24BE, (string)S20LE, (strin
g)S20BE, (string)U20LE, (string)U20BE, (string)S18LE, (string)S18BE, (string)U18LE, (string)U18BE, (string)S16LE, (string)S16BE, (string)U16LE, (strin
g)U16BE, (string)S8, (string)U8 }
                  rate: [ 1, 2147483647 ]
              channels: [ 1, 2147483647 ]
                layout: { (string)interleaved, (string)non-interleaved }

  SRC template: 'src'
    Availability: Always
    Capabilities:
      audio/x-raw
                format: { (string)F64LE, (string)F64BE, (string)F32LE, (string)F32BE, (string)S32LE, (string)S32BE, (string)U32LE, (string)U32BE, (st
ring)S24_32LE, (string)S24_32BE, (string)U24_32LE, (string)U24_32BE, (string)S24LE, (string)S24BE, (string)U24LE, (string)U24BE, (string)S20LE, (strin
g)S20BE, (string)U20LE, (string)U20BE, (string)S18LE, (string)S18BE, (string)U18LE, (string)U18BE, (string)S16LE, (string)S16BE, (string)U16LE, (strin
g)U16BE, (string)S8, (string)U8 }
                  rate: [ 1, 2147483647 ]
              channels: [ 1, 2147483647 ]
                layout: { (string)interleaved, (string)non-interleaved }
```

Properties: Here properties are nothing but "element properties". Each GStreamer Element has its own set of properties that can be manipulated to control the element's operation.

```
Element Properties:
  dithering           : Selects between different dithering methods.
                        flags: readable, writable
                        Enum "GstAudioDitherMethod" Default: 2, "tpdf"
                           (0): none             - GST_AUDIO_DITHER_NONE
                           (1): rpdf             - GST_AUDIO_DITHER_RPDF
                           (2): tpdf             - GST_AUDIO_DITHER_TPDF
                           (3): tpdf-hf          - GST_AUDIO_DITHER_TPDF_HF
  mix-matrix          : Transformation matrix for input/output channels
                        flags: readable, writable
                        GstValueArray of GValues of type "GstValueArray"
  name                : The name of the object
                        flags: readable, writable, 0x2000
                        String. Default: "audioconvert0"
  noise-shaping       : Selects between different noise shaping methods.
                        flags: readable, writable
                        Enum "GstAudioNoiseShapingMethod" Default: 0, "none"
                           (0): none             - GST_AUDIO_NOISE_SHAPING_NONE
                           (1): error-feedback   - GST_AUDIO_NOISE_SHAPING_ERROR_FEEDBACK
                           (2): simple           - GST_AUDIO_NOISE_SHAPING_SIMPLE
                           (3): medium           - GST_AUDIO_NOISE_SHAPING_MEDIUM
                           (4): high             - GST_AUDIO_NOISE_SHAPING_HIGH
  parent              : The parent of the object
                        flags: readable, writable, 0x2000
                        Object of type "GstObject"
  qos                 : Handle Quality-of-Service events
                        flags: readable, writable
                        Boolean. Default: false
(END)
```

5. How does the querying information is passed when the funtion calls are invoked?

Ans: We can query directly the pipeline, by using function **gst_element_query()**. In this we pass "pipeline" as the first argument, the element which we want to query. The query information is stored in GstQuery.

In the pipeline, first the queries are sent to the sinks, and "dispatched" backwards until one element can handle it; that result is sent back to the function caller.

6. Who will send data information into bus?

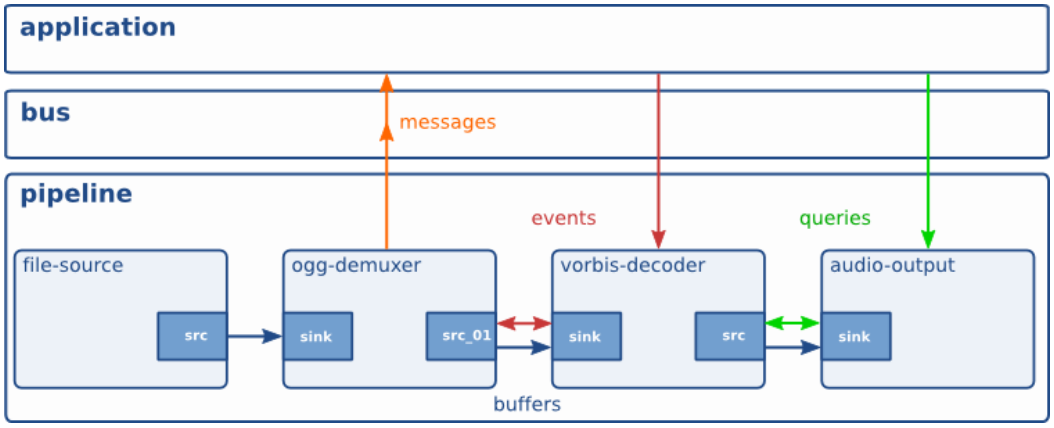
Ans: Elements in the GStreamer pipeline produce messages to communicate with the application. Messages can include information about state changes, end-of-stream (EOS) events, errors, warnings and more.

When an element gives a message, the message is sent to the pipeline's bus. The bus acts as a central message distribution mechanism, this bus collects messages from all elements. To listen the messages on the bus, we use some functions, '**gst_bus_add_watch**' is one among them. When a message arrives on the bus, the application's bus watcher callback is triggered.

In the application we use GST_MESSAGE_TYPE macro, we give the message which we received in the bus as the argument to it. Then this macro gives, which type of error message it is.

7. Block diagram of the data flow in the pipeline.

Ans:



8. What are clock providers? Which elements create clocks?

Ans: Clock providers are nothing but elements which provide GstClock object. Here are some of the scenarios where 'GstClock' objects are created.

Pipeline creation: When you create a pipeline, a default clock is automatically associated with the pipeline.

Element creation: We have certain GStreamer elements, such as audiotestsrc and videotestsrc which create a clock during their initialization.

External Clocks: In some other cases we may create GstClock object and set it on an element or a pipeline, using gst_system_clock_obtain().

Basically clock provider is an element that serves as a source of timing information for the entire pipeline. Clock providers are crucial for maintaining temporal consistency, especially in scenarios involving audio and video playback.

Key points about clock providers in GStreamer:

GStreamer maintains a clock heirarchy within a pipeline. At the top of the heirarchy is the pipeline clock, and beneath it are the clocks of the individual elements. The clock provider is the element that provides the pipeline's master clock. This master clock is used as the primary clock for the entire pipeline.

GStreamer supports different types of clocks, such as system clocks, net clocks and custom clocks. Elements in a pipeline can request the pipeline to use their clock as the master clock. This is typically done by using gst_element_set_clock().

9. gst_debug_bin_to_dot_file_with_ts()

Ans: Above function is a part of Gstreamer's debugging tools, specifically designed for creating Graphviz DOT files that represent the structure and state of the GStreamer pipeline at a specific point in time.

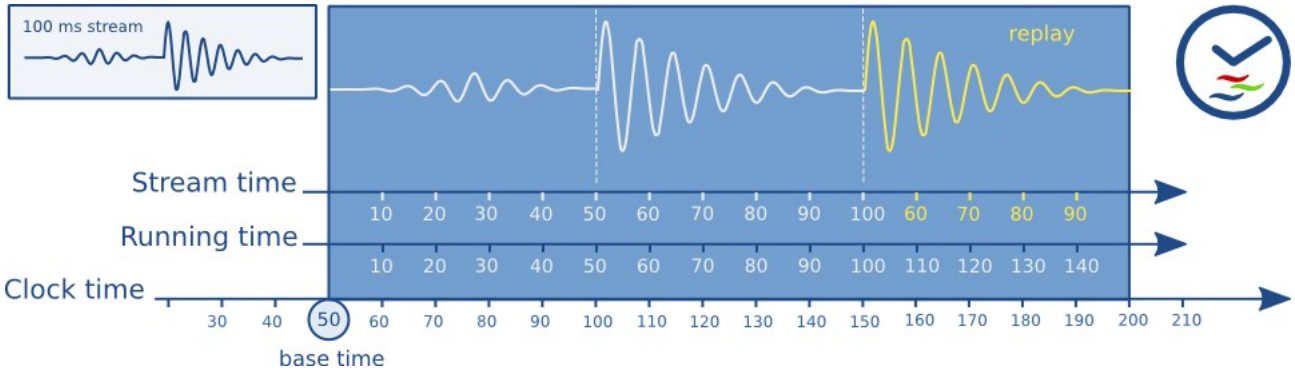
When we use the above function, after running the application a dot file is generated. To see it in the png form, we must run the following command:

- **$dot -Tpng 0.0000dotfile -o pngname.png**

Before running our application we must first export the path using the following command and then compile and run the application:

- **$export GST_DEBUG_DUMP_DOT_DIR=$(pwd)**

10. Explain the time overview along with various time lines used in GStreamer.
Ans:



Clock running-time: **running-time = absolute-time - base-time**
Basically pipeline maintains a GstClock object. This GstClock returns **absolute-time** according to the clock with gst_clock_get_time(). The absolute-time of the previous snapshot is known as **base-time**, this base time is the reference time point used to calculate the current time for scheduling tasks within the pipeline. Thus running-time is the difference between two of them.

The pipeleine maintains GstClock object as well as the base-time when it goes to the PLAYING state. This pipeleine gives the handle to the selected GstClock to each elment along with the base-time. As all objects in the pipeline have the same clock and base-time, we can calculate the running-time according to the pipeline clock.

Buffer running-time: To calculate the buffer running-time, we need buffer timestamp and the preceded SEGMENT event. This event should be converted into GstSegment object. Then we can use the function gst_segment_to_running_time() for calculating the buffer running-time. For non-live sources timestamp buffers with a running-time starting from 0. Where as, live sources need to timestamp buffers with a running-time matching the pipeline running-time.

Buffer stream-time: Here buffer stream-time is nothing but the position in the stream, it is a value between the 0 and the total duration of the media. This is also obtained by buffer timestamps and the preceding SEGMENT event.

Time overview: From the above time overview graph we can understand the following:

- Buffer running-time always increments equally with the clock-time.
- Buffers are played when their running-time is equal to the clock-time – base-time.
- Where as the stream-time represents the position in the stream.

11. Did you worked with your element, which you have created using plugin development steps?

Ans: Yes, I have created "my_filter" element using the plugin development steps and worked on an application. It accepts any type of data and acts as the medium to pass data information.

Here is the output when I have compiled and runned the application for the audio playing using "decodebin" element.

```
sarojini@Softnautics:~/Sarojini_Nama/GStreamer$ gcc audio_decodebin.c `pkg-config --cflags --libs gstreamer-1.0`
sarojini@Softnautics:~/Sarojini_Nama/GStreamer$ ./a.out sample_audio_1.mp3
the pad name :src_0
Linked Audio pad.
I'm plugged, therefore I'm in.
I'm plugged, therefore I'm in.
I'm plugged, therefore I'm in.
I'm plugged, therefore I'm in.
I'm plugged, therefore I'm in.
I'm plugged, therefore I'm in.
I'm plugged, therefore I'm in.
I'm plugged, therefore I'm in.
I'm plugged, therefore I'm in.
I'm plugged, therefore I'm in.
I'm plugged, therefore I'm in.
```

12. Two function calls which I want to explain along with the arguments from my code.

Ans: Function calls:

- **GstPad* gst_element_get_static_pad(element, pad type);** :- This function is used to retrieve a specific static pad of a GStreamer element. Here **element** is the GStreamer element from which to get the pad and **pad type** is nothing but the name of the pad to retrieve. Basically in understandable way we can say that this function provides reference to the existing static pad.

```
GstPad *sinkpad = gst_element_get_static_pad(audio_converter, "sink");
```

- **g_signal_connect(instance, detailed_signal, c_handler, data);** :- This function is used to connect a callback function to a signal emitted by a GObject. It allows you to associate function with a specific event or signal, so that when that event occurs , the connected function is invoked. In this we have **instance** as first argument i.e., nothing but the instance/element to connect, second argument is the **detailed_signal**, which means a string to form signal-name, third argument is the **c_handler**, nothing but a Gcallback to connect and the final argument is data i.e., the data to pass to **c_handler** calls.
  - In my code I have mostly used "pad-added" signal, this refers to a signal emitted when a new pad is added to a GStreamer element.
  - The "pad-added" signal is typically used in dynamic pipelines where elements are added or removed dynamically at runtime.

```
g_signal_connect(demuxer, "pad-added", G_CALLBACK(on_pad_added), NULL);
g_signal_connect(audio_decoder, "pad-added", G_CALLBACK(on_decode_audio_pad_added),NULL);
g_signal_connect(video_decoder, "pad-added", G_CALLBACK(on_decode_video_pad_added),NULL);
```