V4L2: A Status Update

Hans Verkuil
Cisco Systems Norway

Hardware Codec Support

Stateful Codec Support

- Detailed decoder and encoder specification are being written. They describe how to use the V4L2 API for stateful encoders and decoders, including information on how to handle seeks and mid-stream resolution changes. Google (Chromium/ChromeOS team) is the main contributor for this.
- Through the Outreachy organization I am co-mentoring (together with Collabora's Helen Koike) Dafna Hirschfeld who is now working on making the virtual codec driver (vicodec) compliant with these specifications.
- Work is planned to add stateful codec compliance tests as well.

Stateless Codec Support

- Detailed specifications on how to use the V4L2 API for stateless encoders and decoders, including information on how to handle seeks and mid-stream resolution changes, is being written. Google (Chromium/ChromeOS team) is the main contributor for this.
- Dafna Hirschfeld is working on adding a stateless codec support to the existing virtual codec driver (vicodec).
- Work is planned to add stateless codec compliance tests as well.
- Stateless codecs use the new Request API framework. The first stateless MPEG-2 Allwinner SoC decoder was merged as a staging driver in 4.20.
- Work is progressing for Rockchip and Tegra stateless codec support as well, and also for H.264 and HEVC codecs.

Userspace API for Stateless Codecs

- Two device nodes are involved: a /dev/mediaX device is used to create Request objects and the /dev/videoX device is used to stream buffers to/from the codec.
- Userspace is responsible for parsing the headers of the bitstream (for a decoder) or inserting the headers (for an encoder).
- Create a Request object (usually one for every allocated buffer):

```
ioctl(media_fd, MEDIA_IOC_REQUEST_ALLOC, &request_fd);
```

Userspace API for Stateless Codecs

- Queue the output buffer containing the compressed data (for the decoder) or the uncompressed data (for the encoder) of a single frame to the Request object. Tag the output buffer and obtain the reference.

```
struct v4l2_buffer out_buf;
...
out_buf.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
out_buf.request_fd = request_fd;
out_buf.flags = V4L2_BUF_FLAG_REQUEST_FD;
out_buf.timestamp.tv_sec = buf_tag;
out_buf.timestamp.tv_usec = 0;
__u64 buf_ref_ts = v4l2_timeval_to_ns(&out_buf.timestamp);
ioctl(video_fd, VIDIOC_QBUF, &out_buf);
```

Userspace API for Stateless Codecs

- Add the state information to the Request object:

```
struct v4l2_ctrl_mpeg2_slice_params params;
struct v4l2_ext_control ctrl = {};
struct v4l2_ext_controls ctrls = {};

// fill in params
params.backward_ref_ts = some_buf_ref_ts;
ctrl.id = V4L2_CID_MPEG_VIDEO_MPEG2_SLICE_PARAMS;
ctrl.ptr = &params;
ctrl.size = sizeof(param);
ctrls.controls = &ctrl;
ctrls.count = 1;
ctrls.which = V4L2_CTRL_WHICH_REQUEST_VAL;
ctrls.request_fd = request_fd;
ioctl(video_fd, VIDIOC_S_EXT_CTRLS, &ctrls);
```

Userspace API for Stateless Codecs

- Queue the capture buffer that will contain the result:

```
ioctl(video_fd, VIDIOC_QBUF, &cap_buf);
```

- Queue the Request for the output buffer:

```
ioctl(request_fd, MEDIA_REQUEST_IOC_QUEUE, NULL);
```

- Wait for an event from request_fd signaling that the request has completed.
- Dequeue the buffers:

```
ioctl(video_fd, VIDIOC_DQBUF, &out_buf);
ioctl(video_fd, VIDIOC_DQBUF, &cap_buf);
```

- `cap_buf.timestamp == out_buf.timestamp` and `buf_ref_ts` refers to this capture buffer.

Testing

Virtual Drivers

- vivid: emulates webcam/TV/S-Video/HDMI video/vbi capture/output, radio receivers/modulator, software defined receiver, HDMI CEC.
- vim2m: emulates simple video memory-to-memory processing device (such as an RGB to YUV converter).
- vimc: emulates a complex camera pipeline.
- vicodec: emulates a stateful (and soon also a stateless) codec.

Testing with Virtual Drivers

- We want to have better quality control over core media changes in order to avoid regressions. Created a new test-media script to do regression tests for V4L2 and CEC. The plan is to add this to the kernel CI tests (kernelci.org).
- Good compliance tests also make it much easier to write high-quality drivers.
- Developing a new API, the documentation, and testing the new API using virtual drivers is very useful in finding corner cases in the API that you missed.
- syzbot/syzkaller: fuzzing checker from Google. Uses the virtual drivers for testing.

Resources

- Linux Media Infrastructure API: https://linuxtv.org/docs.php
- Version with Codec specification: https://hverkuil.home.xs4all.nl/codec-api/
- Upstream media git repository: http://git.linuxtv.org/media_tree.git
- Cedrus test utility for the stateless decoder: https://github.com/bootlin/v4l2-request-test
- v4l-utils git repository: http://git.linuxtv.org/v4l-utils.git
- linux-media mailinglist & irc channel: http://linuxtv.org/lists.php
- https://linuxtv.org/wiki/index.php/Media_Open_Source_Projects:_Looking_for_Volunteers
- email: hverkuil@xs4all.nl

Questions?