Analysis of the v4l2 library

Video4Linux 2 (V4L2) is a multimedia framework for controlling video devices (e.g., webcams, video capture devices, etc.) on the Linux operating system. V4L2 is an API provided by the Linux kernel, serving as an interface between user applications and hardware.

We analyze one of the wrapped V4L2(in c++) for a better understanding of the V4L2 library used in our project.
<vl42c using videodev2.h>https://github.com/mpromonet/libv4l2cpp/
<videodev2.h>https://github.com/torvalds/linux/blob/master/include/uapi/linux/videodev2.h#L648

1. Camera Device Module (`V4l2Device`)

This class includes the hardware settings of webcam, settings of videos and how streaming is conducted. The abstraction allows for the implementation of both Read/Write and MMap methods for handling video data. This class is controlled by the V4L2Access class, and not used by itself.

1-1. V4L2DeviceParameters (Device Setting)

Device : m_devName

Device name(path) of camera.

Resolution : m_width, m_height

Resolution of the video.

Frame : m_fps

Frame per second.

Video Format : m_formatList

Supported video format from the device.

I/O Type(MMap, ReadWrite) : m_iotype

How video data will be handled. This framework supports MMap and ReadWrite.

1-2. V4L2Device class

virtual bool isReady()

Retrieve whether the device is capable.

virtual bool start()

A method that starts capturing through the camera. This will be re-implemented after being inherited from the appropriate class corresponding to the I/O type.

virtual bool stop()

A method that stops capturing through the camera. This will be re-implemented after being inherited from the appropriate class corresponding to the I/O type.

unsigned int getBufferSize()

Retrieves the buffer size of the video.

unsigned int getFormat()

Retrieves the format of the video.

unsigned int getWidth()

Retrieves the width of resolution.

unsigned int getHeight()

Retrieves the height of resolution

2. Function expansion (V4l2Access)

It is a class for interacting with a device.

We can **get information** such as **File Descriptor, Buffer Size, Format(Pixel type), Width(resolution), Height(resolution)** from the **device currently connected**.

2-1. Method for setting device

void queryFormat()

Querying information related to the format that the device supports. It calls **#VIDIOC_ENUM_FMT**(enum type about format) YUYV, MJPEG, H264 etc..) inside method.

int setFormat(unsigned int format, unsigned int width, unsigned int height)

This method is used for setting format and resolution of V4L2 device. It calls **#VIDIOC_S_FMT**(User designate format will be used in driver)inside the method.

int setFps(int fps)

This method is used for setting the fps speed of the device.

#VIDIOC_S_PARM is used for setting streaming parameters used in the video device.

ex) fps speed of video device, set buffer, time interval etc..

2-2. Method for streaming

int isReady()

It checks whether the device is available or not.

int start()

For Stream-ON(more detail explanation is in Choose a way for processing data)

int stop()

For Stream-OFF

3. Choose a way for processing data

3-1.`V4l2MmapDevice`: It handles high-speed data by memory mapping(streaming and data write/read)

3-1-1. Method for streaming

virtual bool start()

-Stream ON: In order to start streaming, call **#VIDIOC_DQBUF** requests buffers for memory mapping. And then each buffer gets mapped by using **mmap(). Call .#VIDIOC_QBUF to push every buffer in the queue. And call #VIDIOC_STREAMON to start streaming.**

virtual bool stop()

-Stream OFF: Call **#VIDIOC_STREAMOFF** to stop streaming. And delete memory of all buffers that was mapped before by using **mummap()**. And then call **#VIDIOC_REQBUFS** to delete buffers.

3-1-2. Method for writing or reading data

size_t readInternal(char* buffer, size_t bufferSize)

Read data from the V4L2 buffer and copy it to the buffer that the user provides. Call **#VIDIOC_DQBUF to bring dequeued buffer(already used)**.

And then copy data to the buffer that the user provides by using memcpy(). Call **#VIDIOC_QBUF** to push the buffer in the queue again**.**

size_t writeInternal(char* buffer, size_t bufferSize)

Write data that the user provides into the V4L2 buffer.

Call #**VIDIOC_DQBUF** to get an empty buffer. Copy user data into the mapped buffer by using memcpy(). And then call **#VIDIOC_QBUF** to push the buffer in the queue again.

3-2. `V4l2ReadWriteDevice`: It handles data by simple reading or writing method.

It is used for reading or writing data at once..

3-1-2. Method for writing or reading data

size_t readInternal(char* buffer, size_t bufferSize)

return ::write(m_fd, buffer, bufferSize);

Read data from device

size_t writeInternal(char* buffer, size_t bufferSize)

return ::read(m_fd, buffer, bufferSize);

Write data in the buffer

4. Capturing Module (`V4l2Capture`)

This module implements and abstracts the capturing of the camera.

bool isReadable()

A method to check whether the data from the camera can be taken.

size_t Read()

Reads the video data from the buffer.

5. Output Module (`V4l2Output`)

This module implements the output process of the video. This shows the video from the V4L2Capture module.

bool write()

A method to show the video data to output.

size_t writePartial()

A method to show the video data to output partially. These three methods are used to handle large-sized video data.

size_t startPartialWrite()

Starts the partial write of video data.

size_t endPartialWrite()

Ends the partial write of video data.