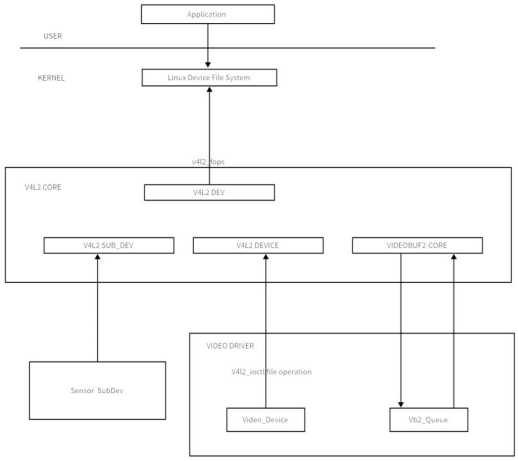
原创 txp玩Linux 于 2022-12-27 21:27:40 发布 56 收藏 版权
文章标签： linux 运维 服务器

一.什么是V4L2框架?

V4L2英文全称是Video for Linux2，它是专门为视频设备设计的内核驱动。在做视频的开发中，一般我们操控V4L2的设备节点就可以直接对摄像头进行操作。通常V4L2在Linux的设备节点是"/dev/video0"。无论是MIPI摄像头还是UVC摄像头，它们底层默认操作的都是/dev/video0的节点。

二. V4L2的代码框架图：



从这张图可以看出来，在使用V4L2进行摄像头操作的时候，都需要访问内核驱动。而整个设备驱动都是由Linux Device File System进行管理，Linux File System管理的是V4L2CORE的功能。而V4L2_CORE里面包含了
V4L2_DEV、V4L2_SUB_DEV、V4L2_DEVICE、VIDEOBUF2_CORE。其中，V4L2_DEIVCE主要是管理视频设备驱动、**VIDEOBUF2_CORE**主要是管理缓冲队列的数据、V4L2_SUB_DEV主要是管理视频设备的子系统。而应用层若想对整个V4L2驱动层进行控制的话，只需要对v4l2_dev进行fops(文件形式控制)操作即可，因为v4l2_dev是驱动层对用户层提供的接口。

三. V4L2代码开发流程：

3.1.打开设备节点：

```
fd = open("/dev/video0", O_RDWR);
if (fd < 0)
{
    printf("open \"%s\" error\n", dev);
    return -1;
}
```

打开/dev/video0视频设备节点

3.2.查询设备的能力

```
struct v4l2_capability cap;
int ret = ioctl(fd, VIDIOC_QUERYCAP, &cap);
if (ret < 0)
{
    printf("VIDIOC_QUERYCAP error\n");
    return -1;
}
printf("driver : %s\n", cap.driver);
printf("device : %s\n", cap.card);
printf("bus : %s\n", cap.bus_info);
printf("version : %d\n", cap.version);
if (cap.capabilities & V4L2_BUF_TYPE_VIDEO_CAPTURE)
{ /*判断是否为视频捕获设备*/
    if (cap.capabilities & V4L2_CAP_STREAMING)
    { /*判断是否支持视频流捕获*/
        printf("support capture\n");
    }
    else
    {
        printf("unsupport capture\n");
    }
}
else
{
    printf("error\n");
    return -1;
}
```

利用ioctl函数访问V4L2的底层命令VIDIOC_QUERYCAP主要是查询摄像头的性能属性。

3.3.获取摄像头支持的格式，并进行像素格式设置

```
struct v4l2_fmtdesc fmtdesc;
fmtdesc.index = 0;
fmtdesc.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
while (ioctl(fd, VIDIOC_ENUM_FMT, &fmtdesc) != -1)
{
    printf("\t%d.%s\n", fmtdesc.index + 1, fmtdesc.description);
    fmtdesc.index++;
}

struct v4l2_format fmt;
fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE; //摄像头缓冲
fmt.fmt.pix.width = 640;
fmt.fmt.pix.height = 480;
fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUV422P;
if (ioctl(fd, VIDIOC_S_FMT, &fmt) < 0)
{
    printf("set foramt:V4L2_PIX_FMT_MJPEG failed\n");
    return -1;
}
```

通过ioctl VIDEOIOC_ENUM_FMT获取对应的支持格式，并且对摄像头进行相应格式的设置。包括：width、height、pixelformat等等。设置完成之后，再使用ioctl命令进行使能操作，使能对应的关键字：**VIDIOC_S_FMT。**

3.4.内存映射到用户空间并进行队列操作

```
int mmap_v4l2_buffer()
{

    usr_buf = (struct USER_BUFFER *)calloc(buf_num, sizeof(struct USER_BUFFER));
    if (!usr_buf)
    {
        printf("calloc \"frame buffer\" error : Out of memory\n");
        return -1;
    }

    struct v4l2_requestbuffers req;
    req.count = buf_num;                    //帧缓冲数量
    req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE; //视频捕获缓冲区类型
    req.memory = V4L2_MEMORY_MMAP;          //内存映射方式
    if (ioctl(fd, VIDIOC_REQBUFS, &req) < 0)
    {
        printf("VIDIOC_REQBUFS fail\n");
        return -1;
    }

    /*映射内核缓存区到用户空间缓冲区*/
```

```
    for (unsigned int i = 0; i < buf_num; ++i)
    {
        /*查询内核缓冲区信息*/
        struct v4l2_buffer v4l2_buf;
        memset(&v4l2_buf, 0, sizeof(v4l2_buf));
        v4l2_buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        v4l2_buf.memory = V4L2_MEMORY_MMAP;
        v4l2_buf.index = i;
        if (ioctl(fd, VIDIOC_QUERYBUF, &v4l2_buf) < 0)
        {
            printf("VIDIOC_QUERYBUF failed\n");
            return -1;
        }

        usr_buf[i].length = v4l2_buf.length;
        usr_buf[i].start = (char *)mmap(0, v4l2_buf.length, PROT_READ | PROT_WRITE, MAP_SHARED, fd, v4l2_buf.m.offset);
        if (MAP_FAILED == usr_buf[i].start)
        { //若映射失败,打印错误
            printf("mmap failed: %d\n", i);
            return -1;
        }
        else
        {
            if (ioctl(fd, VIDIOC_QBUF, &v4l2_buf) < 0)
            { // 若映射成功则将内核缓冲区入队
                printf("VIDIOC_QBUF failed\n");
                return -1;
            }
        }
    }
    return 0;
}
```

利用ioctl控制VIDIOC_REQBUFS，进行视频缓冲区的申请。申请完成之后，把驱动的缓冲区映射到用户空间，映射的api使用的是mmap。若映射用户空间成功的话，则把视频数据入到缓冲区队列，入队对应的ioctl命令是**VIDIOC_QBUF**。

3.5.开启摄像头获取视频流

```
int stream_on()
{
    enum v4l2_buf_type type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    if (ioctl(fd, VIDIOC_STREAMON, &type) < 0)
    {
        printf("VIDIOC_STREAMON failed\n");
        return -1;
    }
    return 0;
}
```

使用stream_on指令使能v4l2摄像头，让它正常启动采集流程。

3.6.从缓冲区中把视频数据取出

```
int write_video_frame()
{
    struct v4l2_buffer v4l2_buf;
    v4l2_buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    v4l2_buf.memory = V4L2_MEMORY_MMAP;
    if (ioctl(fd, VIDIOC_DQBUF, &v4l2_buf) < 0) // 内核缓冲区出队列
    {
        printf("VIDIOC_DQBUF failed, dropped frame\n");
        return -1;
    }
    char buffer[256];
    sprintf(buffer, "%d.mjpg", v4l2_buf.index);
    int file_fd = open(buffer, O_RDWR | O_CREAT); // 若打开失败则不存储该帧图像
    if (file_fd > 0)
    {
        printf("saving %d images\n", v4l2_buf.index);
        write(file_fd, usr_buf[v4l2_buf.index].start, v4l2_buf.bytesused);
        close(file_fd);
    }

    if (ioctl(fd, VIDIOC_QBUF, &v4l2_buf) < 0) //缓冲区重新入队
    {
        printf("VIDIOC_QBUF failed, dropped frame\n");
        return -1;
    }
    return v4l2_buf.index;
}
```

利用ioctl把缓冲区的视频数据取出，它对应的命令是**VIDIOC_DQBUF**。使用了VIDIOC_DQBUF之后，v4l2_buffer结构体就有对应的视频数据了。并把对应的数据写到mjpg图片。做完上述所有操作之后，再把摄像头的数据进行入队操作VIDIOC_QBUF。

3.7.关闭摄像头获取流

```
int camera_stream_off()
{
    /*关闭视频流*/
    enum v4l2_buf_type type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    if (ioctl(fd, VIDIOC_STREAMOFF, &type) == -1)
    {
        printf("Fail to ioctl 'VIDIOC_STREAMOFF'");
        return -1;
    }
    return 0;
}
```

通过ioctl操控指令VIDIOC_STREAMOFF，关闭摄像头采集工作。

3.8.解除映射缓冲区

```
int unmap_v4l2_buffer()
{
    /*解除内核缓冲区到用户缓冲区的映射*/
    for (unsigned int i = 0; i < buf_num; i++)
    {
        int ret = munmap(usr_buf[i].start, usr_buf[i].length);
        if (ret < 0)
        {
            printf("munmap failed\n");
            return -1;
        }
    }
    free(usr_buf); // 释放用户缓冲区内存
    return 0;
}
```

使用munmap api解决内核缓冲区到用户缓冲区的映射操作。

3.9.释放所有的资源

```
void release_camera()
{
    close(fd);
}
```

四.整个工程的运行流程：

```
int main(int argc, char *argv[])
{
    int ret = init_camera("/dev/video0");
    if (ret < 0)
    {
        printf("init_camera error\n");
        return -1;
    }

    ret = mmap_v4l2_buffer();
    if (ret < 0)
    {
        printf("mmap_buffer error\n");
        return -1;
    }

    ret = camera_stream_on();
    if (ret < 0)
    {
        printf("stream_on error\n");
        return -1;
    }

    for (int i = 0; i < 20; i++)
    {
        write_frame();
    }

    ret = camera_stream_off();
    if (ret < 0)
    {
        printf("stream_off error\n");
        return -1;
    }

    ret = unmap_v4l2_buffer();
    if (ret < 0)
```

```
    {
        printf("unmap_buffer error\n");
        return -1;
    }

    release_camera();

    return 0;
}
```

最后：

对嵌入式音频开发感兴趣的朋友，可以加入星球来学习：

最新视频教程更新到：

最后感谢大家的支持哈，加入了星球的朋友，记得加我微信：tu18879499804,我拉大家进解答群！

txp玩Linux
码龄5年　暂无认证

113	1万+	1万+	7万+	
原创	周排名	总排名	访问	等级
1519	119	66	81	285
积分	粉丝	获赞	评论	收藏

私信　关注