OpenBMC development obmc-ikvm code architecture

原创 Lemon in the Rain Modified on 2025-04-17 20:50:20 Read 1.0k Collection 28 Likes 12

Category Column: OpenBMC Article Tags: Architecture linux C++ server

1. obmc-ikmv file

Create two instances of ikvm::Args and ikvm::Manager, and then run the manager manager.run()

cpp AI generated projects 登录复制 run

```cpp
int main(int argc, char* argv[])
{
    // 解析命令行参数
    ikvm::Args args(argc, argv);
    // 创建管理器实例，传入参数
    ikvm::Manager manager(args);

    // 运行管理器
    manager.run();

    // 主函数返回值，表示程序正常退出
    return 0;
}
```

收起 ^

2. ikvm_args file

2.1 Header Files

Defines several variables such as struct CommandLine, frameRate, subsampling, udcName, videoPath, calcFrameCRC, commandLine, etc.

cpp AI generated projects 登录复制 run

```cpp
namespace ikvm
{
class Args
{
  public:

    struct CommandLine
    {
        CommandLine(int c, char** v) : argc(c), argv(v) {}
        int argc;
        char** argv;
    };
  Args(int argc, char* argv[]);
  private:
    void printUsage();
    int frameRate;
    /* @brief Desired subsampling (0: 444, 1: 420) */
    int subsampling;
    /* @brief Path to the USB keyboard device */
    std::string keyboardPath;
    /* @brief Path to the USB mouse device */
    std::string pointerPath;
    /* @brief Name of UDC */
    std::string udcName;
    /* @brief Path to the V4L2 video device */
    std::string videoPath;
    /* @brief Identical frames detection */
    bool calcFrameCRC;
    /* @brief Original command line arguments passed to the application */
    CommandLine commandLine;
};

} // namespace ikvm
```

收起 ^

2.2 Source Files

When creating an object in the main function: ikvm::Args args(argc, argv)

frameRate = 30, subsampling = 0, calcFrameCRC = false, argc and argv are stored in struct commandLine

Struct option is a standard structure definition , and the four data definitions are: long option name, whether to require parameters, flags, short option name

Usage of getopt_long : There are two cases: whether the flag is equal to 0 or not. See the following code explanation, and then refer to AI code explanation for details.

cpp AI generated projects 登录复制 run

```cpp
Args::Args(int argc, char* argv[]) :
    frameRate(30), subsampling(0), calcFrameCRC{false}, commandLine(argc, argv)
{
    int option;
    const char* opts = "f:s:h:k:p:u:v:c";
    struct option lopts[] = {           // 长选项名称、是否需要参数、标志、短选项名称
        {"frameRate", 1, 0, 'f'},       // 设置帧率的选项
        {"subsampling", 1, 0, 's'},     // 设置子采样的选项: 1 = YUV420, 0 = YUV444
        {"help", 0, 0, 'h'},            // 帮助选项
        {"keyboard", 1, 0, 'k'},        // 指定键盘设备路径的选项: /dev/hidg0
        {"mouse", 1, 0, 'p'},           // 指定鼠标设备路径的选项: /dev/hidg1
        {"udcName", 1, 0, 'u'},         // 指定UDC名称的选项: 1e6a0000.usb-vhub:pX
```

```
        {"videoDevice", 1, 0, 'v'},     // 指定视频设备路径的选项：/dev/video0
        {"calcCRC", 0, 0, 'c'},         // 启用CRC计算的选项
        {0, 0, 0, 0}};                  // 选项结束
    // /usr/bin/obmc-ikvm -v /dev/video0 -k /dev/hidg0 -p /dev/hidg1
    // 当标志Buff[2]=0时,option返回值为Buff【3】的值，当flag=1时，option返回值为Buff【2】的值
    while ((option = getopt_long(argc, argv, opts, lopts, NULL)) != -1)
    {
        switch (option)
        {
            case 'f': //optarg是getopt_long函数解析到的参数值，是 <getopt.h> 头文件中定义的一个 extern char *optarg 变量
                frameRate = (int)strtol(optarg, NULL, 0);  // 解析帧率
                if (frameRate < 0 || frameRate > 60)       // 验证帧率范围
                    frameRate = 30;                        // 如果超出范围则重置为默认值
                break;
            case 's':
                subsampling = (int)strtol(optarg, NULL, 0);  // 解析子采样
                if (subsampling < 0 || subsampling > 1)      // 验证子采样范围
                    subsampling = 0;                         // 如果超出范围则重置为默认值
                break;
            case 'h':
                printUsage();  // 打印使用信息
                exit(0);       // 打印帮助信息后退出程序
            case 'k':
                keyboardPath = std::string(optarg);  // 设置键盘设备路径
                break;
            case 'p':
                pointerPath = std::string(optarg);   // 设置鼠标设备路径
                break;
            case 'u':
                udcName = std::string(optarg);       // 设置UDC名称
                break;
            case 'v':
                videoPath = std::string(optarg);     // 设置视频设备路径
                break;
            case 'c':
                calcFrameCRC = true;                 // 启用CRC计算
                break;
        }
    }
}
```

收起 ^

3. ikvm_manager file

3.1 Header Files

The variables and input, video, and server objects are defined as follows. The following points should be noted:

The explicit Manager (const Args & args) keyword is used to improve code security and prevent implicit conversion. The main application scenario is in single-parameter constructors, for example:

Args args;

Manager m(args); //Display call to constructor, correct

Manager m = args; ** // Implicitly calling the constructor, error. Specifically, if a constructor has only one parameter, the compiler may automatically implicitly convert the parameter type to the type of the current class.

cpp AI generated projects 登录复制 run

```cpp
namespace ikvm
{
class Manager
{
  public:
    explicit Manager(const Args& args);
    ~Manager() = default;
    Manager(const Manager&) = default;
    Manager& operator=(const Manager&) = default;
    Manager(Manager&&) = default;
    Manager& operator=(Manager&&) = default;

    /* @brief Begins operation of the VNC server */
    void run();

  private:
    static void serverThread(Manager* manager);
    /* @brief Notifies thread waiters that RFB operations are complete */
    void setServerDone();
    /* @brief Notifies thread waiters that video operations are complete */
    void setVideoDone();
    /* @brief Blocks until RFB operations complete */
    void waitServer();
    /* @brief Blocks until video operations are complete */
    void waitVideo();

    bool continueExecuting;
    /* @brief Boolean to indicate that RFB operations are complete */
    bool serverDone;
    /* @brief Boolean to indicate that video operations are complete */
    bool videoDone;
    /* @brief Input object */
    Input input;
    /* @brief Video object */
    Video video;
    /* @brief RFB server object */
    Server server;
    /* @brief Condition variable to enable waiting for thread completion */
    std::condition_variable sync;
    /* @brief Mutex for waiting on condition variable safely */
    std::mutex lock;
};
```

```
} // namespace ikvm
```

收起 ^

3.2 Source Files

Create a Manager object: **Initialize input, video, and server objects**

continueExecuting = true, serverDone = false, videoDone = true

Assume that: ExecStart=/usr/bin/obmc-ikvm -v /dev/video0 -k /dev/hidg0 -p /dev/hidg1

getKeyboardPath = /dev/hidg0

getPointerPath = /dev/hidg1

getVideoPath = /dev/video0

cpp AI generated projects 登录复制 run

```cpp
Manager::Manager(const Args& args) :
    continueExecuting(true), serverDone(false), videoDone(true),
    input(args.getKeyboardPath(), args.getPointerPath(), args.getUdcName()),
    video(args.getVideoPath(), input, args.getFrameRate(),
          args.getSubsampling()),
    server(args, input, video)
{}
```

Execution Manager Manager.run()

continueExecuting is set to true when initialized

server.wantsFrame() comes from the member function inline bool wantsFrame() const{return server->clientHead ;} in ikvm_server.hpp. As long as there is a kvm session, the value is not NULL.

cpp AI generated projects 登录复制 run

```cpp
void Manager::run()
{
    // 启动服务器线程，执行 serverThread 函数
    std::thread run(serverThread, this);

    // 主循环，持续运行直到 continueExecuting 为 false
    while (continueExecuting)
    {
        // 检查服务器是否需要新的帧
        if (server.wantsFrame())
        {
            // 启动视频设备
            video.start();
            // 获取视频帧
            video.getFrame();
            // 发送帧到服务器
            server.sendFrame();
        }
        else
        {
            // 如果不需要帧，则停止视频设备
            video.stop();
        }

        // 检查视频是否需要调整分辨率
        if (video.needsResize())
        {
            // 等待服务器线程完成当前操作
            waitServer();
            // 标记视频操作未完成
            videoDone = false;
            // 调整视频分辨率
            video.resize();
            // 调整服务器分辨率
            server.resize();
            // 标记视频操作完成，并通知等待的线程
            setVideoDone();
        }
        else
        {
            // 标记视频操作完成，并通知等待的线程
            setVideoDone();
            // 等待服务器线程完成当前操作
            waitServer();
        }
    }

    // 等待服务器线程结束
    run.join();
}
```

收起 ^

Create the thread serverThread() and execute the ****server.run() **function. The run function is very important and is the beginning of calling the libvncserver library rfbProcessEvents() processing function**

cpp AI generated projects 登录复制 run

```cpp
void Manager::serverThread(Manager* manager)
{
    while (manager->continueExecuting)
    {
        manager->server.run();
        manager->setServerDone();
        manager->waitVideo();


```

```
    }
}
```

4. ikvm_server file

4.1 Header Files

Input& input; Video& video; are member variables of reference type, passed through server(args, input, video)

cpp AI generated projects 登录复制 run

```cpp
namespace ikvm
{
class Server
{
  public:

    struct ClientData
    {
        ClientData(int s, Input* i) : skipFrame(s), input(i), last_crc{-1}
        {
            needUpdate = false;
        }

        int skipFrame;
        Input* input;
        bool needUpdate;
        int64_t last_crc;
    };

    Server(const Args& args, Input& i, Video& v);
    ~Server();
    Server(const Server&) = default;
    Server& operator=(const Server&) = default;
    Server(Server&&) = default;
    Server& operator=(Server&&) = default;

    void resize();
    void run();
    void sendFrame();

    inline bool wantsFrame() const
    {
        return server->clientHead;
    }
    inline const Video& getVideo() const
    {
        return video;
    }

  private:
    static void clientFramebufferUpdateRequest(
        rfbClientPtr cl, rfbFramebufferUpdateRequestMsg* furMsg);
    static void clientGone(rfbClientPtr cl);
    static enum rfbNewClientAction newClient(rfbClientPtr cl);
    void doResize();

    bool pendingResize;
    int frameCounter;
    unsigned int numClients;
    long int processTime;
    rfbScreenInfoPtr server;
    Input& input;                   //创建引用，引用ikvm_manager.hpp中的定义
    Video& video;                   //创建引用，引用ikvm_manager.hpp中的定义
    std::vector<char> framebuffer;
    bool calcFrameCRC;
    static constexpr int cursorWidth = 20;
    static constexpr int cursorHeight = 20;

};

} // namespace ikvm
```

收起 ^

4.2 Source Files

The Server object is created in ikvm_manager.hpp, and the server (args, input, video) is initialized when the Manager object is constructed in ikvm_manager.cpp

Resolution adjustment pendingResize = false, frame count frameCounter = 0, session number numClients = 0

Get the important screen buffer structure through rfbScreenInfoPtr server = rfbGetScreen(). This structure is unique and is created when the Server object is created. Later, it will be associated with libvncserver through rfbProcessEvents() in the Server::run() function. For details on the relationship here, see << **The relationship between obmc-ikvm and libvncserver in OpenBMC development** >>

cpp AI generated projects 登录复制 run

```cpp
Server::Server(const Args& args, Input& i, Video& v) :
    pendingResize(false), frameCounter(0), numClients(0), input(i), video(v)
{
    std::string ip("localhost");
    const Args::CommandLine& commandLine = args.getCommandLine();
    int argc = commandLine.argc;
    // ikvm_server.hpp: rfbScreenInfoPtr server
    server = rfbGetScreen(&argc, commandLine.argv, video.getWidth(),
                          video.getHeight(), Video::bitsPerSample,
                          Video::samplesPerPixel, Video::bytesPerPixel);

    if (!server)
    {
        log<level::ERR>("Failed to get VNC screen due to invalid arguments");
        elog<InvalidArgument>(
            xyz::openbmc_project::Common::InvalidArgument::ARGUMENT_NAME(""),

```

```
            xyz::openbmc_project::Common::InvalidArgument::ARGUMENT_VALUE(""));
    }

    framebuffer.resize(
        video.getHeight() * video.getWidth() * Video::bytesPerPixel, 0);

    server->screenData = this;                    //Server对象
    server->desktopName = "OpenBMC IKVM";
    server->frameBuffer = framebuffer.data();     //数据缓冲区
    server->newClientHook = newClient;            //创建新连接时的回调函数，下面详细分析
    server->cursor = rfbMakeXCursor(cursorWidth, cursorHeight, (char*)cursor,
                                    (char*)cursorMask);
    server->cursor->xhot = 1;
    server->cursor->yhot = 1;

    rfbStringToAddr(&ip[0], &server->listenInterface);

    rfbInitServer(server);

    rfbMarkRectAsModified(server, 0, 0, video.getWidth(), video.getHeight());

    server->kbdAddEvent = Input::keyEvent;
    server->ptrAddEvent = Input::pointerEvent;

    processTime = (1000000 / video.getFrameRate()) - 100;

    calcFrameCRC = args.getCalcFrameCRC();
}
```

收起 ^

newClient callback function processing: input.connect/disconnect means that when there is a session, the open source framework configures the USBGadget HID device to enable the keyboard and mouse. When there is no session, the HID UDC is disconnected and the virtual mouse function is disabled.

cpp AI generated projects 登录复制 run

```cpp
enum rfbNewClientAction Server::newClient(rfbClientPtr cl)
{
    Server* server = (Server*)cl->screen->screenData;

    cl->clientData =
        new ClientData(server->video.getFrameRate(), &server->input);           //创建ClientData结构体并赋值到cl->clientData
    cl->clientGoneHook = clientGone;                                            //定义关闭连接回调函数：释放clientData，0会话时input->disconnect()
    cl->clientFramebufferUpdateRequestHook = clientFramebufferUpdateRequest;    //数据帧缓冲区请求回调函数:cd->needUpdate = true
    if (!server->numClients++)                                                  //仅仅在创建第一个会话时，执行input.connect ()
    {
        server->input.connect();
        server->pendingResize = false;
        server->frameCounter = 0;
    }

    return RFB_CLIENT_ACCEPT;
}
```

收起 ^

5. ikvm_video file

5.1 Header Files

Defines several key parameters of the video, such as frame rate, width, height, sampling rate, etc.

Defines the data buffer std::vector buffers. Buffer is a structure with four elements: void* data, bool queued, size_t payload, size_t size

std::vector is a dynamic array container in the C++ standard library. It can store a series of elements of the same type and supports dynamic resizing. In the code, buffers.resize(req.count) is used to dynamically resize the video in the Video::resize() function.

cpp AI generated projects 登录复制 run

```cpp
namespace ikvm
{
class Video
{
  public:
    Video(const std::string& p, Input& input, int fr = 30, int sub = 0);

    char* getData();
    void getFrame();
    bool needsResize();
    void resize();
    void start();
    void stop();
    void restart()
    {
        stop();
        start();
    }

    inline int getFrameRate() const
    {
        return frameRate;
    }
    inline size_t getFrameSize() const
    {
        return buffers[lastFrameIndex].payload;
    }
    inline size_t getHeight() const
    {
        return height;
    }
    inline uint32_t getPixelformat() const
    {

```

```
        return pixelformat;
    }
    inline size_t getWidth() const
    {
        return width;
    }
    inline int getSubsampling() const
    {
        return subSampling;
    }
    inline void setSubsampling(int _sub)
    {
        subSampling = _sub;
    }


    static const int bitsPerSample;
    static const int bytesPerPixel;
    static const int samplesPerPixel;

  private:

    struct Buffer
    {
        Buffer() : data(nullptr), queued(false), payload(0), size(0) {}
        void* data;
        bool queued;
        size_t payload;
        size_t size;
    };


    bool resizeAfterOpen;
    bool timingsError;
    int fd;
    int frameRate;
    int lastFrameIndex;
    size_t height;
    size_t width;
    int subSampling;
    Input& input;
    const std::string path;
    std::vector<Buffer> buffers;
    uint32_t pixelformat;
};

} // namespace ikvm
```

收起 ^

5.2 Source Files

Create a Video object and initialize it: resolution 800*600, sampling rate subSampling = args.getSubsampling() initialization value 0, path = /dev/video0

Several key functions are defined: Video::getData(), Video::getFrame(), bool Video::needsResize(), void Video::resize(), void Video::start(), void Video::stop()

cpp AI generated projects 登录复制 run

```
Video::Video(const std::string& p, Input& input, int fr, int sub) :
    resizeAfterOpen(false), timingsError(false), fd(-1), frameRate(fr),
    lastFrameIndex(-1), height(600), width(800), subSampling(sub), input(input),
    path(p), pixelformat(V4L2_PIX_FMT_JPEG)
{}
```

Video::start(), a few points to note are:

When executed for the first time, all codes are executed completely and the global handle fd is created. The subsequent execution will determine whether /dev/video0 is open by judging fd. If it is open, there is no need to set it again.

Then, the following operations are performed: wake up the input device, set the frame rate, set the sampling mode YUV420/444, adjust the video stream size, and adjust the resolution

cpp AI generated projects 登录复制 run

```
/**
 * @brief 启动视频设备并进行初始化
 *
 * 本函数负责打开视频设备，查询并设置设备的参数，如分辨率和帧率等。
 * 它还负责调整视频流的格式和参数，以满足应用程序的需求。
 */
void Video::start()
{
    int rc;
    size_t oldHeight = height;
    size_t oldWidth = width;
    v4l2_capability cap;
    v4l2_format fmt;
    v4l2_streamparm sparm;
    v4l2_control ctrl;

    // 如果文件描述符fd大于等于0，则表明设备已经打开，直接返回
    if (fd >= 0)
    {
        return;
    }

    // 发送唤醒数据包以激活输入设备
    input.sendWakeupPacket();

    // 打开视频设备文件
    fd = open(path.c_str(), O_RDWR);
    if (fd < 0)
    {

```

```
        // 如果打开设备失败，记录错误日志并抛出异常
        log<level::ERR>("Failed to open video device",
                        entry("PATH=%s", path.c_str()),
                        entry("ERROR=%s", strerror(errno)));
        elog<Open>(
            xyz::openbmc_project::Common::File::Open::ERRNO(errno),
            xyz::openbmc_project::Common::File::Open::PATH(path.c_str()));
    }

    // 查询视频设备的能力
    memset(&cap, 0, sizeof(v4l2_capability));
    rc = ioctl(fd, VIDIOC_QUERYCAP, &cap);
    if (rc < 0)
    {
        // 如果查询设备能力失败，记录错误日志并抛出异常
        log<level::ERR>("Failed to query video device capabilities",
                        entry("ERROR=%s", strerror(errno)));
        elog<ReadFailure>(
            xyz::openbmc_project::Common::Device::ReadFailure::CALLOUT_ERRNO(
                errno),
            xyz::openbmc_project::Common::Device::ReadFailure::CALLOUT_DEVICE_PATH(path.c_str()));
    }

    // 检查视频设备是否支持视频捕获和流式传输
    if (!(cap.capabilities & V4L2_CAP_VIDEO_CAPTURE) ||
        !(cap.capabilities & V4L2_CAP_STREAMING))
    {
        // 如果不支持，记录错误日志并抛出异常
        log<level::ERR>("Video device doesn't support this application");
        elog<Open>(
            xyz::openbmc_project::Common::File::Open::ERRNO(errno),
            xyz::openbmc_project::Common::File::Open::PATH(path.c_str()));
    }

    // 查询视频设备的格式
    memset(&fmt, 0, sizeof(v4l2_format));
    fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    rc = ioctl(fd, VIDIOC_G_FMT, &fmt);
    if (rc < 0)
    {
        // 如果查询设备格式失败，记录错误日志并抛出异常
        log<level::ERR>("Failed to query video device format",
                        entry("ERROR=%s", strerror(errno)));
        elog<ReadFailure>(
            xyz::openbmc_project::Common::Device::ReadFailure::CALLOUT_ERRNO(
                errno),
            xyz::openbmc_project::Common::Device::ReadFailure::CALLOUT_DEVICE_PATH(path.c_str()));
    }

    // 设置视频设备的帧率参数
    memset(&sparm, 0, sizeof(v4l2_streamparm));
    sparm.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    sparm.parm.capture.timeperframe.numerator = 1;
    sparm.parm.capture.timeperframe.denominator = frameRate;
    rc = ioctl(fd, VIDIOC_S_PARM, &sparm);
    if (rc < 0)
    {
        // 如果设置帧率失败，记录警告日志
        log<level::WARNING>("Failed to set video device frame rate",
                            entry("ERROR=%s", strerror(errno)));
    }

    // 设置视频设备的JPEG色度抽样方式
    ctrl.id = V4L2_CID_JPEG_CHROMA_SUBSAMPLING;
    ctrl.value = subSampling ? V4L2_JPEG_CHROMA_SUBSAMPLING_420
                             : V4L2_JPEG_CHROMA_SUBSAMPLING_444;
    rc = ioctl(fd, VIDIOC_S_CTRL, &ctrl);
    if (rc < 0)
    {
        // 如果设置JPEG色度抽样失败，记录警告日志
        log<level::WARNING>("Failed to set video jpeg subsampling",
                            entry("ERROR=%s", strerror(errno)));
    }

    // 更新视频设备的分辨率和像素格式
    height = fmt.fmt.pix.height;
    width = fmt.fmt.pix.width;
    pixelformat = fmt.fmt.pix.pixelformat;

    // 检查是否支持像素格式
    if (pixelformat != V4L2_PIX_FMT_RGB24 && pixelformat != V4L2_PIX_FMT_JPEG)
    {
        // 如果不支持，记录错误日志
        log<level::ERR>("Pixel Format not supported",
                        entry("PIXELFORMAT=%d", pixelformat));
    }

    // 调整视频流的大小
    resize();

    // 检查分辨率是否发生变化
    if (oldHeight != height || oldWidth != width)
    {
        resizeAfterOpen = true;
    }
}
```

收起 ^

`munmap` Used to cancel a memory mapping previously `mmap` created by a system call. It removes the specified memory area from the process's address space.

cpp AI generated projects 登录复制 run

```cpp
// 停止视频流并释放相关资源
void Video::stop()
{
    int rc;
    unsigned int i;
    v4l2_buf_type type(V4L2_BUF_TYPE_VIDEO_CAPTURE);

    // 如果文件描述符fd小于0，则不执行停止操作，直接返回
    if (fd < 0)
    {
        return;
    }

    // 重置最后帧的索引
    lastFrameIndex = -1;

    // 停止视频流的捕获
    rc = ioctl(fd, VIDIOC_STREAMOFF, &type);
    if (rc)
    {
        // 如果停止失败，记录错误日志
        log<level::ERR>("Failed to stop streaming",
                        entry("ERROR=%s", strerror(errno)));
    }

    // 释放所有缓冲区资源
    for (i = 0; i < buffers.size(); ++i)
    {
        // 如果缓冲区已分配内存，则释放内存并重置相关标志
        if (buffers[i].data)
        {
            munmap(buffers[i].data, buffers[i].size);
            buffers[i].data = nullptr;
            buffers[i].queued = false;
        }
    }

    // 关闭文件描述符
    close(fd);
    fd = -1;
}
```

收起 ^

Video::getFrame() obtains the underlying Driver data and stores it in the data buffer std::vector buffers. This function mainly performs the following operations

1. Check the validity of the device file descriptor
2. Set file descriptor set and timeout
3. Switch to non-blocking mode to avoid permanent blocking
4. Poll the device to obtain valid frame data
5. Process the acquired video buffer (**VIDIOC_DQBUF: Get data to send to the front-end display**)
6. Requeue unused buffers (**VIDIOC_QBUF: Requeue buffer logic, join video capture queue**)

There are two particularly important points to note here:

IOCTRL:VIDIOC_DQBUF

- Purpose: **Take out a processed buffer from the queue of the video device so that the application can access the data in it** .
- Usage scenario: Usually during video capture, the application needs to take out a buffer filled with data from the queue in order to process or display the data.

IOCTRL:VIDIOC_QBUF

- Purpose: **Add the buffer to the queue of the video device so that the device can start processing the data in the buffer** .
- Usage scenario: Usually during video capture or output, the application needs to queue a buffer so that the device can fill it with data (capture) or send data (output).

The purpose of temporarily switching to non-blocking mode:

The code first uses `fcntl(fd, F_SETFL, fd_flags | O_NONBLOCK)` to set the file descriptor `fd` to non-blocking mode. The main purpose of doing this is to `select` implement a **timeout mechanism** with the function . `select` The function will wait for the file descriptor to be readable (data arrives) or time out. If the file descriptor is in blocking mode and no data arrives, `select` it will block forever, which is exactly the situation mentioned in the code comment "to prevent the driver from being permanently blocked when the video signal is lost."

Reasons for reverting to blocking mode:

After `select` the call, the code immediately restores `fcntl(fd, F_SETFL, fd_flags)` the file descriptor `fd` to its original flags using `fd_flags` . After temporarily modifying the attributes of a file descriptor, it should be restored to its original state to avoid unexpected effects on other parts of the program.

In summary, the code temporarily sets the file descriptor to non-blocking mode in order to `select` implement a timeout mechanism in the call to prevent the program from blocking forever when there is no video signal. `select` After the call ends, in order to maintain the original behavior of the file descriptor and avoid affecting other potential operations, the code will restore it to the previous blocking mode (or other original mode).

cpp AI generated projects 登录复制 run

```cpp
void Video::getFrame()
{
    int rc(0); // 定义返回码变量，用于存储函数调用的结果
    int fd_flags; // 定义文件描述符标志变量，用于保存原始的文件描述符状态
    v4l2_buffer buf; // 定义V4L2缓冲区结构体变量，用于存储从设备获取的视频帧信息
    fd_set fds; // 定义文件描述符集合变量，用于select系统调用，监控文件描述符的可读性
    timeval tv; // 定义时间结构体变量，用于select系统调用，设置超时时间

    // 设备有效性检查
    if (fd < 0)
```

```
    {
        return; // 如果文件描述符无效（小于0），则直接返回，表示无法获取视频帧
    }

    // 初始化文件描述符集合
    FD_ZERO(&fds); // 清空文件描述符集合，确保集合中没有任何文件描述符
    FD_SET(fd, &fds); // 将视频设备的文件描述符添加到集合中，select将监控这个文件描述符

    // 设置select超时时间为1秒
    tv.tv_sec = 1; // 设置秒部分为1秒
twen    tv.tv_usec = 0; // 设置微秒部分为0，总共超时时间为1秒
twen
twen    // 初始化V4L2缓冲区结构
twen    memset(&buf, 0, sizeof(v4l2_buffer)); // 将缓冲区结构体清零，确保所有成员都被初始化为0
    buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE; // 设置缓冲区类型为视频捕获
    buf.memory = V4L2_MEMORY_MMAP; // 设置缓冲区内存映射模式为MMAP（内存映射）

    /* 切换非阻塞模式（关键安全措施）：
     * 防止视频信号丢失时驱动程序永久阻塞
     * 保留原始文件描述符标志用于后续恢复 */
    fd_flags = fcntl(fd, F_GETFL); // 获取当前文件描述符的标志（包括阻塞/非阻塞等状态）
    fcntl(fd, F_SETFL, fd_flags | O_NONBLOCK); // 设置文件描述符为非阻塞模式，即使没有数据可读，read/ioctl等操作也会立即返回，避免程序永久等待

    // 等待设备数据就绪（最长等待1秒）
    rc = select(fd + 1, &fds, NULL, NULL, &tv); // 使用select系统调用等待视频设备文件描述符变为可读状态。fd + 1是需要监控的最大文件描述符加1，中间两个NULL表示不监控写和异常情况，tv是超时时间。
    if (rc > 0) // 如果select返回值大于0，表示在超时时间内有文件描述符就绪（这里是视频设备）
    {
        // 循环处理所有就绪缓冲区
        do
        {
            // 从队列中取出缓冲区
            rc = ioctl(fd, VIDIOC_DQBUF, &buf); // 使用ioctl命令VIDIOC_DQBUF从驱动程序的捕获队列中取出一个已经填充好数据的缓冲区。buf中会包含帧数据的信息。
            if (rc >= 0) // 如果ioctl返回值大于等于0，表示成功取出一个缓冲区
            {
                buffers[buf.index].queued = false; // 将该缓冲区标记为未排队，表示它正在被处理

                // 成功获取有效帧处理
                if (!(buf.flags & V4L2_BUF_FLAG_ERROR)) // 检查缓冲区标志，如果未设置错误标志，则认为这是一个有效的视频帧
                {
                    lastFrameIndex = buf.index; // 记录最后一个成功获取的帧的索引
                    buffers[lastFrameIndex].payload = buf.bytesused; // 记录该帧的实际数据大小（有效载荷）
                    break; // 获取到有效帧后，跳出当前的do-while循环，因为我们只需要一个最新的有效帧
                }
                else    // 错误帧处理
                {
                    buffers[buf.index].payload = 0; // 如果是错误帧，则将有效载荷大小设置为0
                }
            }
        } while (rc >= 0); // 持续循环，直到ioctl(VIDIOC_DQBUF)返回错误（表示队列中没有更多已填充的缓冲区）
    }

    // 恢复原始阻塞模式设置
    fcntl(fd, F_SETFL, fd_flags); // 将文件描述符恢复到之前保存的阻塞模式

    /* 缓冲区重新排队逻辑：
     * 1. 跳过当前使用的最后一帧缓冲区
     * 2. 将所有未排队的缓冲区重新加入采集队列
     * 3. 维持环形缓冲区的持续运转 */
    for (unsigned int i = 0; i < buffers.size(); ++i) // 遍历所有已分配的缓冲区
    {
        if (i == (unsigned int)lastFrameIndex) // 如果当前遍历的缓冲区索引是刚刚使用的最后一帧的索引，则跳过，避免重复排队
        {
            continue;
        }

        if (!buffers[i].queued) // 如果当前缓冲区没有被排队（queued标志为false），表示它需要被重新放入采集队列
        {
            // 重新初始化缓冲区结构
            memset(&buf, 0, sizeof(v4l2_buffer)); // 再次清空缓冲区结构体
            buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE; // 设置缓冲区类型为视频捕获
            buf.memory = V4L2_MEMORY_MMAP; // 设置缓冲区内存映射模式为MMAP
            buf.index = i; // 设置缓冲区的索引

            // 将缓冲区加入采集队列
            rc = ioctl(fd, VIDIOC_QBUF, &buf); // 使用ioctl命令VIDIOC_QBUF将缓冲区重新放入驱动程序的捕获队列中，等待驱动程序填充新的视频数据
            if (rc) // 如果ioctl返回值不为0，表示排队操作失败
            {
                log<level::ERR>("Failed to queue buffer", // 记录错误日志
                                entry("ERROR=%s", strerror(errno))); // 包含具体的错误信息（通过strerror获取）
            }
            else // 如果排队操作成功
            {
                buffers[i].queued = true; // 将该缓冲区标记为已排队
            }
        }
    }
}
```

收起 ^

One thing that needs special attention is that when i == **lastFrameIndex** , buffers[i].queued remains unchanged, which means that it is not added to the acquisition queue, and the data read by Video::getData() is exactly the data of this frame.

buffers[lastFrameIndex].data This data will be sent to the KVM front end for rendering and display via Server::sendFrame()

buffers[buf.index].queued = false Do not add to the underlying acquisition queue for now, frame data that needs to be processed by the application layer

buffers[buf.index].queued = true Join the underlying acquisition queue and prepare to obtain the underlying Video data frame

cpp AI generated projects 登录复制 run

```cpp
char* Video::getData()
{
    if (lastFrameIndex >= 0)
    {
        return (char*)buffers[lastFrameIndex].data;
    }

    return nullptr;
}
```

6. ikvm_input file

6.1 Header Files

Defines several important variables such as keyboardFd, pointerFd, etc.

cpp AI generated projects 登录复制 run

```cpp
namespace ikvm
{
class Input
{
  public:
    Input(const std::string& kbdPath, const std::string& ptrPath,
          const std::string& udc);

    void connect();
    void disconnect();
    static void keyEvent(rfbBool down, rfbKeySym key, rfbClientPtr cl);
    static void pointerEvent(int buttonMask, int x, int y, rfbClientPtr cl);
    void sendWakeupPacket();

  private:
    static constexpr int NUM_MODIFIER_BITS = 4;
    static constexpr int KEY_REPORT_LENGTH = 8;
    static constexpr int PTR_REPORT_LENGTH = 6;

    static constexpr const char* hidUdcPath =
        "/sys/kernel/config/usb_gadget/obmc_hid/UDC";
    static constexpr const char* usbVirtualHubPath =
        "/sys/bus/platform/devices/1e6a0000.usb-vhub";
    static constexpr int HID_REPORT_RETRY_MAX = 5;

    static uint8_t keyToMod(rfbKeySym key);
    static uint8_t keyToScancode(rfbKeySym key);
    bool writeKeyboard(const uint8_t* report);
    void writePointer(const uint8_t* report);

    int keyboardFd;
    int pointerFd;
    uint8_t keyboardReport[KEY_REPORT_LENGTH];
    uint8_t pointerReport[PTR_REPORT_LENGTH];
    std::string keyboardPath;
    std::string pointerPath;
    std::string udcName;
    std::map<int, int> keysDown;
    std::ofstream hidUdcStream;
    std::mutex keyMutex;
    std::mutex ptrMutex;
};

} // namespace ikvm
```

收起 ^

6.2 Source Files

The Input object is created in the ikvm_manager.cpp file. The specific constructor is as follows:

The object defines a std::ofstream hidUdcStream type object for file output operations

cpp AI generated projects 登录复制 run

```cpp
Input::Input(const std::string& kbdPath, const std::string& ptrPath,
             const std::string& udc) :
    keyboardFd(-1),
    pointerFd(-1), keyboardReport{0}, pointerReport{0}, keyboardPath(kbdPath),
    pointerPath(ptrPath), udcName(udc)
{
    hidUdcStream.exceptions(std::ofstream::failbit | std::ofstream::badbit);
    hidUdcStream.open(hidUdcPath, std::ios::out | std::ios::app);
}
```

In addition, several important functions are defined, among which keyEvent and pointerEvent are registered to libvncserver through callback functions when creating the Server object.

rfbScreenInfoPtr rfbScreen->kbdAddEvent = Input::keyEvent;

rfbScreenInfoPtr rfbScreen->kbdAddEvent = Input::pointerEvent;

The last point to note is that keyEvent also involves key value conversion, that is, the local keyboard or virtual keyboard key value ASCII captured by the KVM front end is sent here via WSS, then converted into standard 8-byte HID data, and finally sent to the Host via USB.

cpp AI generated projects 登录复制 run

```cpp
void Input::connect();          //使能HID虚拟设备: echo "1e6a0000.usb-vhub:p6">/sys/kernel/config/usb_gadget/obmc_hid/UDC
void Input::disconnect();       //断开HID虚拟设备: echo " ">/sys/kernel/config/usb_gadget/obmc_hid/UDC
void Input::keyEvent(rfbBool down, rfbKeySym key, rfbClientPtr cl);
void Input::pointerEvent(int buttonMask, int x, int y, rfbClientPtr cl);
bool Input::writeKeyboard(const uint8_t* report);  //键值发送函数
void Input::writePointer(const uint8_t* report);   //鼠标发送函数
```