The relationship between obmc-ikvm and libvncserver in OpenBMC development

原创 Lemon in the Rain Posted on 2025-04-19 15:56:46 Views: 558 Collection 17 Likes 7

Category Column: OpenBMC Article Tags: database

1. System startup process

- Start the service (obmc-ikvm.service): ExecStart=/usr/bin/obmc-ikvm -v /dev/video0 -k /dev/hidg0 -p /dev/hidg1
- ikvm_server.cpp/mian() function creates Args and Manager objects, parses input parameters and creates management service ikvm::Manager manager(args)
- When ikvm_manager.cpp creates the Manager object, it initializes the input/video/server objects. The three objects are created in the ikvm_manager.hpp header file.

c AI generated projects 登录复制 run

```c
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)            //创建Server对象
{}
void Manager::run()
{
    std::thread run(serverThread, this);
    ..........
}

void Manager::serverThread(Manager* manager)
{
    while (manager->continueExecuting)
    {
        manager->server.run();              //执行Server.run()函数
        manager->setServerDone();
        manager->waitVideo();
    }
}
```

收起 ^

- When creating a Server object, the vncserver screen buffer structure rfbScreenInfoPtr **rfbScree** is created through rfbGetScreen() , and the Server::run() function calls **rfbProcessEvents()**

cpp AI generated projects 登录复制 run

```cpp
rfbScreenInfoPtr server;
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(),          //在libvncserver中calloc内存，并对结构体初始化赋值
                          video.getHeight(), Video::bitsPerSample,            //代码位置：src/libvncserver/main.c
                          Video::samplesPerPixel, Video::bytesPerPixel);
    ..............
}

void Server::run()
{   // libvncserver/main.c/L1261/rfbBool rfbProcessEvents(rfbScreenInfoPtr screen,long usec)
    rfbProcessEvents(server, processTime); // 与libvncserver相关的函数、处理事件从这里开始
                                           // 代码位置：src/libvncserver/main.c
    if (server->clientHead)
    {
        frameCounter++;
        if (pendingResize && frameCounter > video.getFrameRate())
        {
            doResize();
            pendingResize = false;
        }
    }
}
```

收起 ^

- libvncserver/mian.c/rfbProcessEvents() is the entrance of the entire libvncserver. The key to socket monitoring and message processing is the rfbCheckFds(screen, usec) function.

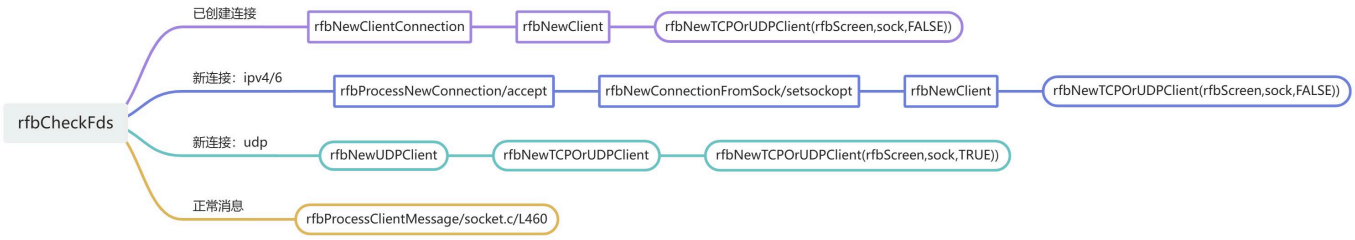
c AI generated projects 登录复制 run

```c
//obmc-ikvm函数入口：Server::run()
rfbBool rfbProcessEvents(rfbScreenInfoPtr screen,long usec)
{
  rfbClientIteratorPtr i;
  rfbClientPtr cl,clPrev;
  rfbBool result=FALSE;
  extern rfbClientIteratorPtr
    rfbGetClientIteratorWithClosed(rfbScreenInfoPtr rfbScreen);

  /* 如果未指定超时时间，使用屏幕的延迟更新时间 */
  if(usec<0)
    usec=screen->deferUpdateTime*1000;
```

```
  /* 检查文件描述符集合中的活动，处理网络 I/O 事件 */
  rfbCheckFds(screen,usec);

  /* 检查 HTTP 文件描述符集合中的活动，处理 HTTP 请求 */
  rfbHttpCheckFds(screen);

  /* 获取包含已关闭连接的客户端迭代器 */
  i = rfbGetClientIteratorWithClosed(screen);

  /* 遍历所有客户端连接 */
  cl=rfbClientIteratorHead(i);
  while(cl) {
    /* 更新客户端屏幕 */
    result = rfbUpdateClient(cl);

    /* 保存当前客户端指针，以便在迭代器前进后检查连接状态 */
    clPrev=cl;
    cl=rfbClientIteratorNext(i);

    /* 如果客户端套接字无效，表示连接已关闭 */
    if(clPrev->sock==RFB_INVALID_SOCKET) {
      /* 处理客户端连接断开事件 */
      rfbClientConnectionGone(clPrev);
      result=TRUE;
    }
  }

  /* 释放客户端迭代器 */
  rfbReleaseClientIterator(i);

  return result;
}
```

收起 ^

- **rfbCheckFds mainly does two things: monitor new socket connections and forward normal messages (rfbHttpCheckFds monitors Http connections and has similar functions)**



- The **rfbNewTCPOrUDPClient()** function is where rfbClientPtr cl and iterator rfbClientIteratorPtr are initialized, and it is also where the newClientHook() hook function is called (the callback is defined in ikvm_server.cpp enum rfbNewClientAction Server::newClient(rfbClientPtr cl))

2. Client Iterator

2.1 Dynamic Linked List

cpp　　AI generated projects　　登录复制　　run

```cpp
//libvncserver/src/libvncserver/rfbserver.c
struct rfbClientIterator
{
     rfbClientPtr next;
     rfbScreenInfoPtr screen;
     rfbBool closedToo;
}
struct rfbClientIterator;
//libvncserver/include/rfb/rfb.h
//屏幕帧缓冲区结构体
typedef struct _rfbScreenInfo
{
    void* screenData;                    //ikvm_server.cpp中定义：Server对象
    char* frameBuffer;                   //ikvm_server.cpp中定义：宽*高*bytesPerPixel

    struct _rfbClientRec* clientHead;    //存储最后一个建立会话的cl信息
    struct _rfbClientRec* pointerClient;  /**< "Mutex" for pointer events */
    ..................
} rfbScreenInfo, *rfbScreenInfoPtr;
//libvncserver/include/rfb/rfb.h
//连接到VNC服务器的客户端信息
typedef struct _rfbClientRec {
    rfbScreenInfoPtr screen;
    void* clientData;
    struct _rfbClientRec *prev;           //创建的最后cl为NULL，其他的见下图
    struct _rfbClientRec *next;           //创建的首个cl为NULL，其他的见下图
    ..................
} rfbClientRec, *rfbClientPtr;
```

收起 ^

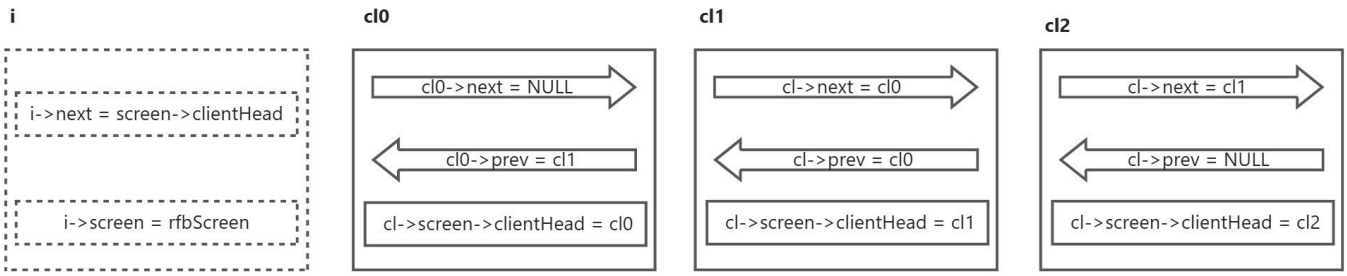
If the definition is: rfbClientIterator i = rfbGetClientIterator(screen), 3 KVM sessions are started: rfbClientPtr cl0 cl1 cl2

- **rfbScreen is created when the obmc-ikvm: Server object is created and is unique.**
- **Server.run() -> fbProcessEvents(server, processTime) -> i->screen = rfbScreen, i->next = screen->clientHead [libvncserver/main.c/fbProcessEvents(L1278)]**
- **CL is a two-way dynamic linked list, and its assignment logic is: libvncserver/src/libvncserver/rfbserver.c/rfbNewTCPOrUDPClient(L470)**

c AI generated projects 登录复制 run

```c
//libvncserver/src/libvncserver/rfbserver.c/rfbNewTCPOrUDPClient(L470)
static rfbClientPtr
rfbNewTCPOrUDPClient(rfbScreenInfoPtr rfbScreen,
                     rfbSocket sock,
                     rfbBool isUDP)
{
    .................
    cl->next = rfbScreen->clientHead;      //首个会话为clientHead = NULL，指向前一个cl
    cl->prev = NULL;
    if (rfbScreen->clientHead)
        rfbScreen->clientHead->prev = cl; //修改前一个cl的prev值

    rfbScreen->clientHead = cl;           //永远存储最后一个创建的会话cl
    .................
}
```

收起 ^



2.2 Initialization

The first initialization is completed in libvncserver/main.c/rfbProcessEvents() function. The specific scenario is:

c AI generated projects 登录复制 run

```c

  i = rfbGetClientIteratorWithClosed(screen);              //malloc迭代器结构体，初始化赋值i->screen = rfbScreen
  cl=rfbClientIteratorHead(i);                             //获取最后建立的客户端，cl = i->next = i->screen->clientHead

  while(cl) {
    /* 更新客户端屏幕 */
    result = rfbUpdateClient(cl);

    /* 保存当前客户端指针，以便在迭代器前进后检查连接状态 */
    clPrev=cl;
    cl=rfbClientIteratorNext(i);                           //动态链表查询，直到查询到第一个建立的客户端，其cl0->next = NULL
    。。。。。
    }
```

收起 ^

2.3 Application Scenarios

- Create and get an iterator: it = rfbGetClientIterator(server)
- Dynamic linked list query client: cl = rfbClientIteratorNext(it), when the first client is found, cl0->next = NULL, thus exiting