简约的博客

极简而约

个人资料

极简科技

关注 发私信

访问： 46751次

积分： 958

等级： BLOG 3

排名： 千里之外

原创： 48篇 转载： 0篇

译文： 2篇 评论： 5条

Android ION机制_HAL与vendor层共享内存_流程简介(1)

标签： android 架构 内存

2016-04-09 12:25 1452人阅读 评论(0) 收藏 举报

分类：

Adroid之Camera开发（15）

研究导向：

open camera过程中有些初始化设置参数需要从vendor层获取，而vendor与hal隶属于不同进程，通过ION机制设置共享内存来实现不同进程间数据共享，下面简要介绍流程：

在初始化过程中hal层会通过socket将消息发送至vendor层-通知vendor map共享内存。

ION初始化过程：

vim hardware/qcom/camera/QCamera2/HAL/QCamera2HWI.cpp

```
int QCamera2HardwareInterface::openCamera()
{
  ......
  if (NULL == gCamCapability[mCameraId]) {
    if(NO_ERROR != initCapabilities(mCameraId,mCameraHandle)) {  //会初始化ION
      ......
    }
  }
  mCameraHandle->ops->register_event_notify(mCameraHandle->camera_handle,
                                            camEvtHandle,
                                            (void *) this);
  ......
  pthread_mutex_lock(&m_parm_lock);/* MM-MC-FixString8HeapCorrupt-00+ */
   //初始化m_pCapability-从vendor层拿到的camera相关初始化数据
  mParameters.init(gCamCapability[mCameraId], mCameraHandle, this, this);
  pthread_mutex_unlock(&m_parm_lock);/* MM-MC-FixString8HeapCorrupt-00+ */
  mParameters.m_parm_lock_ptr = &m_parm_lock;/* MM-MC-FixString8HeapCorrupt-00+ */
  ALOGI("openCamera: m_parm_lock_ptr = 0x%x", mParameters.m_parm_lock_ptr);/* MM-MC-FixString8HeapCorrupt-00+ */
```

```
    mCameraOpened = true;
    gCameraOpened = true;/* MM-CL-CTS-testMultiCameraRelease-00+ */
#ifdef USE_ARCSOFT_FEATURE
    if ((NULL != mArcSoft_Feature)&&(mCameraId == 0))/* MM-MC-FixOpenCameraCrash-00+ */
        mArcSoft_Feature->imx214_module_source = gCamCapability[0]->fih_imx214_module_source;//MM-
YW-Get module source for HAL-00
#endif
    return NO_ERROR;
}
int QCamera2HardwareInterface::initCapabilities(uint32_t cameraId,
        mm_camera_vtbl_t *cameraHandle)
{
    ATRACE_CALL();
    int rc = NO_ERROR;
    QCameraHeapMemory *capabilityHeap = NULL;
    /* Allocate memory for capability buffer */
    capabilityHeap = new QCameraHeapMemory(QCAMERA_ION_USE_CACHE);
    rc = capabilityHeap->allocate(1, sizeof(cam_capability_t)); //1、设置ION共享内存
    ......
    /* Map memory for capability buffer */
memset(DATA_PTR(capabilityHeap,0), 0, sizeof(cam_capability_t));
//2、通过socket通知vendor映射共享内存用于进程间通信
    rc = cameraHandle->ops->map_buf(cameraHandle->camera_handle,
                        CAM_MAPPING_BUF_TYPE_CAPABILITY,
                        capabilityHeap->getFd(0),
                        sizeof(cam_capability_t));
    if(rc < 0) {
        ALOGE("%s: failed to map capability buffer", __func__);
        goto map_failed;
    }
    /* Query Capability */
    rc = cameraHandle->ops->query_capability(cameraHandle->camera_handle);
    if(rc < 0) {
        ALOGE("%s: failed to query capability",__func__);
        goto query_failed;
    }
    gCamCapability[cameraId] = (cam_capability_t *)malloc(sizeof(cam_capability_t));
    if (!gCamCapability[cameraId]) {
        ALOGE("%s: out of memory", __func__);
        goto query_failed;
    }
```

关闭

```
    memcpy(gCamCapability[cameraId], DATA_PTR(capabilityHeap,0),
                                    sizeof(cam_capability_t));
    //copy the preview sizes and video sizes lists because they
    //might be changed later
    copyList(gCamCapability[cameraId]->preview_sizes_tbl, savedSizes[cameraId].all_preview_sizes,
            gCamCapability[cameraId]->preview_sizes_tbl_cnt);
    savedSizes[cameraId].all_preview_sizes_cnt = gCamCapability[cameraId]->preview_sizes_tbl_cnt;
    copyList(gCamCapability[cameraId]->video_sizes_tbl, savedSizes[cameraId].all_video_
            gCamCapability[cameraId]->video_sizes_tbl_cnt);
    savedSizes[cameraId].all_video_sizes_cnt = gCamCapability[cameraId]->video_sizes_t
    rc = NO_ERROR;
query_failed:
    cameraHandle->ops->unmap_buf(cameraHandle->camera_handle,
                        CAM_MAPPING_BUF_TYPE_CAPABILITY);
......
}
```

1、设置ION共享内存：

```
vim hardware/qcom/camera/QCamera2/HAL/QCameraMem.cpp
int QCameraHeapMemory::allocate(uint8_t count, size_t size)
{
    traceLogAllocStart(size, count, "HeapMemsize");
    unsigned int heap_mask = 0x1 << ION_IOMMU_HEAP_ID;
    int rc = alloc(count, size, heap_mask);  //主要填充mMemInfo
    if (rc < 0)
        return rc;
    for (int i = 0; i < count; i ++) {
        void *vaddr = mmap(NULL,   //映射共享内存
                mMemInfo[i].size,
                PROT_READ | PROT_WRITE,
                MAP_SHARED,
                mMemInfo[i].fd, 0);
        if (vaddr == MAP_FAILED) {
            for (int j = i-1; j >= 0; j --) {
                munmap(mPtr[j], mMemInfo[j].size);
                mPtr[j] = NULL;
                deallocOneBuffer(mMemInfo[j]);
            }
            return NO_MEMORY;
        } else
            mPtr[i] = vaddr;
    }
```

关闭

```
    if (rc == 0)
        mBufferCount = count;
    traceLogAllocEnd((size * count));
    return OK;
}
int QCameraMemory::alloc(int count, size_t size, unsigned int heap_id)
{
    ......s
    int new_bufCnt = mBufferCount + count;
    traceLogAllocStart(size, count, "Memsize");
    ......
    for (int i = mBufferCount; i < new_bufCnt; i ++) {
        if ( NULL == mMemoryPool ) {
            CDBG_HIGH("%s : No memory pool available and So allocate new buffer", __func__);
            rc = allocOneBuffer(mMemInfo[i], heap_id, size, m_bCached);
            if (rc < 0) {
                ALOGE("%s: AllocateIonMemory failed", __func__);
                for (int j = i-1; j >= 0; j--)
                    deallocOneBuffer(mMemInfo[j]);
                break;
            }
        } else {
            rc = mMemoryPool->allocateBuffer(mMemInfo[i],
                                             heap_id,
                                             size,
                                             m_bCached,
                                             mStreamType);
            if (rc < 0) {
                ALOGE("%s: Memory pool allocation failed", __func__);
                for (int j = i-1; j >= 0; j--)
                    mMemoryPool->releaseBuffer(mMemInfo[j],
                                               mStreamType);
                break;
            }
        }
    }
    traceLogAllocEnd (size * (size_t)count);
    return rc;
}
int QCameraMemory::allocOneBuffer(QCameraMemInfo &memInfo,
        unsigned int heap_id, size_t size, bool cached)
```

关闭

```
{
    int rc = OK;
    struct ion_handle_data handle_data;
    struct ion_allocation_data alloc;
    struct ion_fd_data ion_info_fd;
    int main_ion_fd = 0;
    main_ion_fd = open("/dev/ion", O_RDONLY);
    if (main_ion_fd < 0) {
        ALOGE("Ion dev open failed: %s\n", strerror(errno));
        goto ION_OPEN_FAILED;
    }
    memset(&alloc, 0, sizeof(alloc));
    memset(&ion_info_fd, 0, sizeof(ion_info_fd));/* MM-MC-ModifyIonFdInitFlow-00+ */

    alloc.len = size;
    /* to make it page size aligned */
    alloc.len = (alloc.len + 4095U) & (~4095U);
    alloc.align = 4096;
    if (cached) {
        alloc.flags = ION_FLAG_CACHED;
    }
    alloc.heap_id_mask = heap_id;
    rc = ioctl(main_ion_fd, ION_IOC_ALLOC, &alloc);
    if (rc < 0) {
        ALOGE("ION allocation failed: %s\n", strerror(errno));
        goto ION_ALLOC_FAILED;
    }
    //memset(&ion_info_fd, 0, sizeof(ion_info_fd));/* MM-MC-ModifyIonFdInitFlow-00- */
    ion_info_fd.handle = alloc.handle;
    rc = ioctl(main_ion_fd, ION_IOC_SHARE, &ion_info_fd);
    if (rc < 0) {
        ALOGE("ION map failed %s\n", strerror(errno));
        goto ION_MAP_FAILED;
    }
    memInfo.main_ion_fd = main_ion_fd;
    memInfo.fd = ion_info_fd.fd;
    memInfo.handle = ion_info_fd.handle;
    memInfo.size = alloc.len;
    memInfo.cached = cached;
    memInfo.heap_id = heap_id;
    CDBG_HIGH("%s : ION buffer %lx with size %d allocated",
```

关闭

```
        __func__, (unsigned long)memInfo.handle, memInfo.size);
    return OK;
ION_MAP_FAILED:
    memset(&handle_data, 0, sizeof(handle_data));
    handle_data.handle = ion_info_fd.handle;
    ioctl(main_ion_fd, ION_IOC_FREE, &handle_data);
ION_ALLOC_FAILED:
    close(main_ion_fd);
ION_OPEN_FAILED:
    return NO_MEMORY;
}
```

2、通过socket通知vendor映射相应共享内存用于进程间通信：

vim hardware/qcom/camera/QCamera2/stack/mm-camera-interface/src/mm_camera_interface.c

```
static int32_t mm_camera_intf_map_buf(uint32_t camera_handle,
                                      uint8_t buf_type,
                                      int fd,
                                      size_t size)
{
    int32_t rc = -1;
    mm_camera_obj_t * my_obj = NULL;
    pthread_mutex_lock(&g_intf_lock);
    my_obj = mm_camera_util_get_camera_by_handler(camera_handle);
    if(my_obj) {
        pthread_mutex_lock(&my_obj->cam_lock);
        pthread_mutex_unlock(&g_intf_lock);
        rc = mm_camera_map_buf(my_obj, buf_type, fd, size);
    } else {
        pthread_mutex_unlock(&g_intf_lock);
    }
    return rc;
}
```

vim hardware/qcom/camera/QCamera2/stack/mm-camera-interface/src/mm_camera.c

```
int32_t mm_camera_map_buf(mm_camera_obj_t *my_obj,
                          uint8_t buf_type,
                          int fd,
                          size_t size)
{
    int32_t rc = 0;
    cam_sock_packet_t packet;
    memset(&packet, 0, sizeof(cam_sock_packet_t));
    packet.msg_type = CAM_MAPPING_TYPE_FD_MAPPING;
```

关闭

```
    packet.payload.buf_map.type = buf_type;
    packet.payload.buf_map.fd = fd;
    packet.payload.buf_map.size = size;
    rc = mm_camera_util_sendmsg(my_obj,
                                &packet,
                                sizeof(cam_sock_packet_t),
                                fd);
    pthread_mutex_unlock(&my_obj->cam_lock);
    return rc;
}
int32_t mm_camera_util_sendmsg(mm_camera_obj_t *my_obj,
                               void *msg,
                               size_t buf_size,
                               int sendfd)
{
    int32_t rc = -1;
    uint32_t status;
    /* need to lock msg_lock, since sendmsg until reposonse back is deemed as one operation*/
    pthread_mutex_lock(&my_obj->msg_lock);
    if(mm_camera_socket_sendmsg(my_obj->ds_fd, msg, buf_size, sendfd) > 0) {
        /* wait for event that mapping/unmapping is done */
        mm_camera_util_wait_for_event(my_obj, CAM_EVENT_TYPE_MAP_UNMAP_DONE, &status);
        if (MSM_CAMERA_STATUS_SUCCESS == status) {
            rc = 0;
        }
    }
    pthread_mutex_unlock(&my_obj->msg_lock);
    return rc;
}
vim hardware/qcom/camera/QCamera2/stack/mm-camera-interface/src/mm_camera_sock.c
int mm_camera_socket_sendmsg(
  int fd,  //socket fd
  void *msg,
  size_t buf_size,
  int sendfd)
{
    struct msghdr msgh;
    struct iovec iov[1];
    struct cmsghdr * cmsghp = NULL;
    char control[CMSG_SPACE(sizeof(int))];
    if (msg == NULL) {
```

关闭

```
    CDBG("%s: msg is NULL", __func__);
    return -1;
  }
  memset(&msgh, 0, sizeof(msgh));
  msgh.msg_name = NULL;
  msgh.msg_namelen = 0;
  iov[0].iov_base = msg;
  iov[0].iov_len = buf_size;
  msgh.msg_iov = iov;
  msgh.msg_iovlen = 1;
  CDBG("%s: iov_len=%llu", __func__,
       (unsigned long long int)iov[0].iov_len);
  msgh.msg_control = NULL;
  msgh.msg_controllen = 0;
  /* if sendfd is valid, we need to pass it through control msg */
  if( sendfd > 0) {
    msgh.msg_control = control;
    msgh.msg_controllen = sizeof(control);
    cmsghp = CMSG_FIRSTHDR(&msgh);
    if (cmsghp != NULL) {
      CDBG("%s: Got ctrl msg pointer", __func__);
      cmsghp->cmsg_level = SOL_SOCKET;
      cmsghp->cmsg_type = SCM_RIGHTS;
      cmsghp->cmsg_len = CMSG_LEN(sizeof(int));
      *((int *)CMSG_DATA(cmsghp)) = sendfd;
      CDBG("%s: cmsg data=%d", __func__, *((int *) CMSG_DATA(cmsghp)));
    } else {
      CDBG("%s: ctrl msg NULL", __func__);
      return -1;
    }
  }
  return sendmsg(fd, &(msgh), 0);  //socket发送数据到vendor通知映射共享内存
}
```

refer: ION机制介绍

http://blog.csdn.net/cosmoslhf/article/details/41209925

- 上一篇 Callback机制_基础（1）

- 下一篇 Android 设置参数至kernel_ois为例

相关文章推荐

- Android ION机制_HAL与vendor层共享内存_流程...
- 轻松拿下Linux进程、线程和调度
- Qualcomm平台camera调试移植入门
- 30天掌握机器学习升级版
- camera video数据流
- Python网络爬虫快速入门实战
- Android Camera HAL V3 Vendor Tag及V1，V3参...
- 最适合自学的C++基础知识
- android之ION内存管理器（1）-- 简介
- 一招学会Android自定义控件
- ION基本概念介绍和原理分析
- 从零练就iOS高手
- PMEM原理分析
- ION基本概念介绍和原理分析
- android之ION内存管理器（1）-- 简
- Android ION机制_HAL与vendor层共

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

关闭