

邮箱: zhunxun@gmail.com

< 2020年5月 >						
日	一	二	三	四	五	六
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

搜索

找找看

谷歌搜索

PostCategories

C语言(2)
IO Virtualization(3)
KVM虚拟化技术(26)
linux 内核源码分析(61)
Linux日常应用(3)
linux时间子系统(3)
qemu(10)
seLinux(1)
windows内核(5)
调试技巧(2)
内存管理(8)
日常技能(3)
容器技术(2)
生活杂谈(1)
网络(5)
文件系统(4)
硬件(4)

PostArchives

2018/4(1)
2018/2(1)
2018/1(3)
2017/12(2)
2017/11(4)
2017/9(3)
2017/8(1)
2017/7(8)
2017/6(6)
2017/5(9)
2017/4(15)
2017/3(5)
2017/2(1)
2016/12(1)
2016/11(11)
2016/10(8)
2016/9(13)

ArticleCategories

时态分析(1)

Recent Comments

- Re:virtio前端驱动详解
我看了下, Linux-4.18.2中的vp_notify()函数。bool vp_notify(struct virtqueue *vq){ /* we write the queue's sele
c...
--Linux-inside
- Re:virtIO之VHOST工作原理简析

QEMU IO事件处理框架

Qemu IO事件处理框架

qemu是基于事件驱动的,在基于KVM的qemu模型中,每一个VCPU对应一个qemu线程,且qemu主线程负责各种事件的监听,这里有一个小的IO监听框架,本节对此进行介绍。

1.1 涉及结构



```
struct GArray {  
  
    gchar *data;  
  
    guint len;  
  
};
```



Data指向一个GpollFD数组, len表示数组的个数。



```
struct GPollFD  
  
{  
  
    gint          fd;  
  
    gushort       events;  
  
    gushort       revents;  
  
};
```



Fd为监听的fd, event为请求监听的事件,是一组bit的组合。Revents为poll收到的事件,根据此判定当前什么事件可用。



```
typedef struct IOHandlerRecord {  
  
    IOCanReadHandler *fd_read_poll;  
  
    IOHandler *fd_read;  
  
    IOHandler *fd_write;  
  
    void *opaque;  
  
    QLIST_ENTRY(IOHandlerRecord) next;  
  
    int fd;  
  
    int pollfds_idx;  
  
    bool deleted;  
  
} IOHandlerRecord;
```



再问一个问题，从设置ioeventfd那个流程来看的话是guest发起一个IO，首先会陷入到kvm中，然后由kvm向qemu发送一个IO到来的event，最后IO才被处理，是这样的吗？

--Linux-inside

3. Re:virtIO之VHOST工作原理简析
你好。设置ioeventfd这个部分和guest里面的virtio前端驱动有关系吗？
设置ioeventfd和virtio前端驱动是如何发生联系起来的？谢谢。

--Linux-inside

4. Re:QEMU IO事件处理框架
良心博主，怎么停跟了，太可惜了。
--黄铁牛
5. Re:linux 逆向映射机制浅析
小哥哥520脱单了么

--黄铁牛

Top Posts

- 1. 详解操作系统中断(21154)
- 2. PCI 设备详解一(15808)
- 3. 进程的挂起、阻塞和睡眠(13714)
- 4. Linux下桥接模式详解一(13467)
- 5. virtio后端驱动详解(10539)

推荐排行榜

- 1. 进程的挂起、阻塞和睡眠(6)
- 2. qemu-kvm内存虚拟化1(2)
- 3. 为何要写博客(2)
- 4. virtIO前后端notify机制详解(2)
- 5. 详解操作系统中断(2)

该结构是qemu中IO框架的处理单位，fd_read和fd_write为注册的对应处理函数。Next表示该结构会连接在一个全局的链表上，fd是对应的fd，delete标志是否需要从链表中删除该结构。


1.2 代码分析

1.2.1 初始化阶段

Main-> qemu_init_main_loop

Main-> main_loop-> main_loop_wait

qemu_init_main_loop



```
int qemu_init_main_loop(void)
{
    int ret;

    GSource *src;

    init_clocks();

    ret = qemu_signal_init();

    if (ret) {

        return ret;
    }

    //malloc a globble fd array

    gpollfds = g_array_new(FALSE, FALSE, sizeof(GPollFD));

    //create a aio context

    qemu_aio_context = aio_context_new();

    //get event source from aio context

    src = aio_get_g_source(qemu_aio_context);

    //add source to main loop

    g_source_attach(src, NULL);

    g_source_unref(src);

    return 0;
}
```




Qemu中的main loop主要采用 了glib中的事件循环，关于此详细内容，准备后面专门写一小节，本节主要看主体IO框架。

该函数主要就分配了一个Garray结构存储全局的GpollFD，在main_loop中的main_loop_wait阶段有两个比较重要的函数：qemu_iohandler_fill，os_host_main_loop_wait和qemu_iohandler_poll，前者把用户添加的fd信息注册到刚才分配的Garray结构中，os_host_main_loop_wait对事件进行监听，qemu_iohandler_poll对接收到的事件进行处理。

1.2.2 添加fd

用户添加fd的函数为qemu_set_fd_handler，参数中fd为本次添加的fd，后面分别是对该fd的处理函数（read or write），最后opaque为处理函数的参数。



```
int qemu_set_fd_handler(int fd,

                        IOHandler *fd_read,
```

```

        IOHandler *fd_write,

        void *opaque)

{

    return qemu_set_fd_handler2(fd, NULL, fd_read, fd_write, opaque);

}

```

可见该函数直接调用了qemu_set_fd_handler2:

```

int qemu_set_fd_handler2(int fd,

        IOCanReadHandler *fd_read_poll,

        IOHandler *fd_read,

        IOHandler *fd_write,

        void *opaque)

{

    IOHandlerRecord *ioh;

    assert(fd >= 0);

    //if read and write are null,delete

    if (!fd_read && !fd_write) {

        QLIST_FOREACH(ioh, &io_handlers, next) {

            if (ioh->fd == fd) {

                ioh->deleted = 1;

                break;

            }

        }

    } else { //find and goto find

        QLIST_FOREACH(ioh, &io_handlers, next) {

            if (ioh->fd == fd)

                goto found;

        }

        ioh = g_malloc0(sizeof(IOHandlerRecord));

        //insert ioh to io_handlers list

        QLIST_INSERT_HEAD(&io_handlers, ioh, next);

    found:

        ioh->fd = fd;

        ioh->fd_read_poll = fd_read_poll;

        ioh->fd_read = fd_read;

        ioh->fd_write = fd_write;

        ioh->opaque = opaque;

        ioh->pollfds_idx = -1;
    }
}

```

```

        ioh->deleted = 0;

        qemu_notify_event();

    }

    return 0;
}

```



这里判断如果read和write函数均为空的话就表示本次是要delete某个fd，就遍历所有的io_handlers，对指定的fd对应的IOHandlerRecord标志delete。

否则还有两种情况，添加或者更新。所以首先还是要从io_handlers找一下，如果找到直接更新，否则新创建一个IOHandlerRecord，然后再添加信息。具体信息内容就比较简单。

1.2.3 处理fd

在main_loop_wait函数中，通过os_host_main_loop_wait对fd进行监听，当然并不是它直接监听，而是通过glib的接口。

当os_host_main_loop_wait返回后，就表示当前有可用的事件，在main_loop_wait函数中，调用了qemu_iohandler_poll函数对fd进行处理。



```

void qemu_iohandler_poll(GArray *pollfds, int ret)
{
    if (ret > 0) {
        IOHandlerRecord *pioh, *ioh;

        QLIST_FOREACH_SAFE(ioh, &io_handlers, next, pioh) {

            int revents = 0;

            if (!ioh->deleted && ioh->pollfds_idx != -1) {
                GPollFD *pfd = &g_array_index(pollfds, GPollFD,
                                                ioh->pollfds_idx);

                revents = pfd->revents;

            }

            if (!ioh->deleted && ioh->fd_read &&
                (revents & (G_IO_IN | G_IO_HUP | G_IO_ERR))) {
                ioh->fd_read(ioh->opaque);
            }

            if (!ioh->deleted && ioh->fd_write &&
                (revents & (G_IO_OUT | G_IO_ERR))) {
                ioh->fd_write(ioh->opaque);
            }

            /* Do this last in case read/write handlers marked it for deletion */
            if (ioh->deleted) {
                QLIST_REMOVE(ioh, next);

                g_free(ioh);
            }
        }
    }
}

```

```
}  
  
}  
  

```

具体的处理倒也简单，逐个遍历io_handlers，对于每个GpollFD，取其revents，判断delete标志并校验状态，根据不同的状态，调用read或者write回调。最后如果是delete的GpollFD，就从链表中remove掉，释放GpollFD。

补充：针对qemu进程中线程数目的问题，从本节可以发现qemu主线程主要负责事件循环，针对每个虚拟机的VCPU，会有一个子线程为之服务，因此qemu线程数目至少要大于等于1+VCPU数目。

以马内利！

参考资料：

1、qemu 2.7源码

分类: [qemu](#)

[好文要顶](#)[关注我](#)[收藏该文](#)



[jack.chen](#)
[关注 - 12](#)
[粉丝 - 44](#)
[+加关注](#)

00

« 上一篇: [vhost-user 分析1](#)

posted @ 2018-04-10 19:45 jack.chen Views(1512) Comments(3) Edit 收藏

Post Comment

#1楼 2018-11-12 09:49 | Wali8822

以马内利!!

支持(0) 反对(0)

#2楼 2019-07-05 15:08 | Jack_JF

楼主写的博客都不错啊，可惜只是更新到了18年，如果能继续往后更新就好了

支持(0) 反对(0)

#3楼 2019-10-16 16:24 | 黄铁牛

良心博主，怎么停跟了，太可惜了。

支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)， [访问](#) 网站首页。

最新 IT 新闻:

- 腾讯在列！微软宣布超140家工作室为Xbox Series X开发游戏
 - 黑客声称从微软GitHub私人数据库当中盗取500GB数据
 - IBM开源用于简化AI模型开发的Elyra工具包
 - 中国网民人均安装63个App：腾讯系一家独大
 - Lyft颁布新规：强制要求乘客和司机佩戴口罩
- » [更多新闻...](#)

历史上的今天:

2017-04-10 Linux进程虚拟地址空间的管理

Copyright © 2020 jack.chen
Powered by .NET Core on Kubernetes

以马内利