2022 虎符 pwn hfdev（二）

原创

yongbaoii 于 2022-03-30 07:30:00 发布 99 收藏

分类专栏： CTF 文章标签： 网络安全

本文链接：https://blog.csdn.net/yongbaoii/article/details/123789641

承接上文

https://blog.csdn.net/yongbaoii/article/details/123789460

BH

那么我们现在就理解了BH在整个qemu中的地位。

说白了就是可以通过bh，能让qemu的poll机制产生回调。

这种中断并不是在qemu中被提出的，是在linux里面就有的。

在一些linux的书籍文献中这样介绍：

中断句柄只是中断处理的第一部分，因为下面一些限制，必须要有另外一部分来完整的处理中断：
1.中断句柄是异步运行的，只要发生中断就会进入中断句柄，而中断的发生是异步的，也就是说随时可以发生。因此中断句柄可能会中断一些正在执行的重要代码，包括另外一个中断句柄。
2.中断句柄运行时需要关中断，最好情况是禁止了当前的中断级（未设置SA_INTERRUPT)，最坏的情况是所有的中断被中断(设置SA_INTERRUPT）。当然，关中断只是运行中断句柄时必要的步骤，适当的时候还是需要打开的。同样这也决定了中断句柄要尽快结束。
3.中断句柄一般都是时间紧迫的任务，因为他们处理的是硬件。
4.中断句柄不运行在进程上下文，因此不能被堵塞。
因此，中断处理分成了两部分，第一部分就是中断句柄(top halves)，一旦发生中断就必须执行；第二部分叫做bottom halves，用来处理剩下的那些不那么紧急的工作，目的很简单，就是让中断句柄尽可能快的返回。

那么我们再来详细看一下qemu中的BH

BH 数据结构

```
struct QEMUBH {
    AioContext *ctx; // 下半部所在的context
    QEMUBHFunc *cb;  // 下半部要执行的函数
    void *opaque;  // 函数参数
    QEMUBH *next;  // 下一个要执行的下半部
    bool scheduled;  // 使能bh，是否被调度，true:下一次dispatch会触发cb； false:下一次dispatch不会触发cb
    bool idle;
    bool deleted;  // 标记是否将bh删除
};
```

BH数据结构的最后三个属性用于控制事件循环线程对BH的操作行为。
为什么要有这三个属性？因为bh的注册和执行是异步地，因此需要有一种方法提供给注册者用来通知执行者，注册者期望bh被怎样执行。
scheduled用来通知执行者，期望在下一次dispatch时bh被调度到；idle用于通知执行者bh不用作progress计数；deleted通知执行者bh被调度一次之后就删除。

qemu主线程有默认的事件循环qemu_aio_context，bh挂载到主线程的事件循环有接口qemu_bh_new可以直接调用

```
/* Functions to operate on the main QEMU AioContext.  */
QEMUBH *qemu_bh_new(QEMUBHFunc *cb, void *opaque)
{
    return aio_bh_new(qemu_aio_context, cb, opaque);
}
```

bh还有一种用法是只执行一次，然后就删除bh，这个通过`aio_bh_schedule_oneshot`接口可以实现
而普通bh执行方式是调用`qemu_bh_schedule`，虽然也只调度一次但不会被删除，下一次调度不需要重新注册

事件循环在poll到fd准备好之后，会调度qemu定制的回调函数`aio_ctx_dispatch`，其主要功能是执行fd对应的回调函数，但在此之前会先检查context上是否挂载有bh，如果有先执行bh。

```
/* 注册到AioContext的回调函数，当fd准备好之后，事件循环机制会调度这个回调 */
static void
aio_dispatch(AioContext *ctx)
{
    qemu_lockcnt_inc(&ctx->list_lock);
    aio_bh_poll(ctx);
    aio_dispatch_handlers(ctx);
    qemu_lockcnt_dec(&ctx->list_lock);
}

static gboolean
aio_ctx_dispatch(GSource     *source,
                 GSourceFunc  callback,
                 gpointer     user_data)
{
#ifdef DEBUG
    g_print("%s\n", __FUNCTION__);
#endif
    AioContext *ctx = (AioContext *) source;

    aio_dispatch(ctx);
    return TRUE;
}
```

qemu时钟系统

qemu支持四种时钟

QEMU_CLOCK_REALTIME 不受虚拟系统的影响，随时间流逝而累加计数
QEMU_CLOCK_VIRTUAL 虚拟时钟，记录虚拟系统的时间滴答
QEMU_CLOCK_HOST 这个类似墙上时钟，修改宿主机系统时间会改变这个时间
QEMU_CLOCK_VIRTUAL_RT 在非icount模式下和QEMU_CLOCK_VIRTUAL，在icount模式下于QEMU_CLOCK_VIRTUAL不同的是在虚拟cpu休眠的时候该值也会累加

四个时钟使用类型作为索引放在qemu_clocks数组中，描述时钟的数据结构为QEMUClock。
QEMUClock数据结构如下

```
typedef struct QEMUClock {
    /* We rely on BQL to protect the timerlists */
    QLIST_HEAD(, QEMUTimerList) timerlists;

    NotifierList reset_notifiers;
    int64_t last;

    QEMUClockType type;
    bool enabled;
} QEMUClock;
```

QEMUTimerList用于存放定时器链表

timerlist_new函数用于初始化时钟的定时器链表

```
QEMUTimerList *timerlist_new(QEMUClockType type,
                             QEMUTimerListNotifyCB *cb,
                             void *opaque)
{
    QEMUTimerList *timer_list;
    QEMUClock *clock = qemu_clock_ptr(type);

    timer_list = g_malloc0(sizeof(QEMUTimerList));
    qemu_event_init(&timer_list->timers_done_ev, true);
    timer_list->clock = clock;
    timer_list->notify_cb = cb;
    timer_list->notify_opaque = opaque;
    qemu_mutex_init(&timer_list->active_timers_lock);
    QLIST_INSERT_HEAD(&clock->timerlists, timer_list, list);
    return timer_list;
}
```

这个函数其实就是把一个定时器回调函数注册到了这个时钟的timer队列里面
主线程的定时器不当可以从定时器结构QEMUClock.timer_list 中索引，还可以从main_loop_tlg中索引。

再介绍一个timer_init_full函数
这是源码

```
void timer_init_full(QEMUTimer *ts,
                     QEMUTimerListGroup *timer_list_group, QEMUClockType type,
                     int scale, int attributes,
                     QEMUTimerCB *cb, void *opaque);

/**
 * timer_init:
 * @ts: the timer to be initialised
 * @type: the clock to associate with the timer
 * @scale: the scale value for the timer
 * @cb: the callback to call when the timer expires
 * @opaque: the opaque pointer to pass to the callback
 *
 * Initialize a timer with the given scale on the default timer list
 * associated with the clock.
 * See timer_init_full for details.
 */
```

看注释就能明白
这个函数对QEMUTimer结构体进行了设置。
设置它所在了timerlist
设置时间
设置回调函数等等。