原有的记录重放技术对pio读操作的记录方式是直接记录读取的数据，改进后的记录方式是记录对应数据在虚拟镜像中的真实物理扇区号

但是，根据实验分析，原有的记录重放技术记录的pio数据绝大部分只有1个字节的长度，

也有这样的情况的

port:1297, direction:0, size:1, count:4

port:1297, direction:0, size:1, count:64

port:1297, direction:0, size:1, count:64

port:1297, direction:0, size:1, count:1024

但是读取的内容也不是从磁盘扇区中读取的，因此log中直接记录读取到的数据

512字节长度的才是真正意义上读取的完整数据，如读取的/boot/文件夹下的某个文件，但是其中也有一些长度为512字节的记录数据并不是从磁盘扇区中读取的，因此现在的策略是，只把长度=512字节,且从磁盘中读取的数据记录方式变为记录对应的磁盘扇区号，其他的还是直接记录读取的数据

目前记录扇区号的情况大概是

port:496, direction:0, size:4, count:128

port:496, direction:0, size:2, count:256

port:368, direction:0, size:4, count:128

port:368, direction:0, size:2, count:256

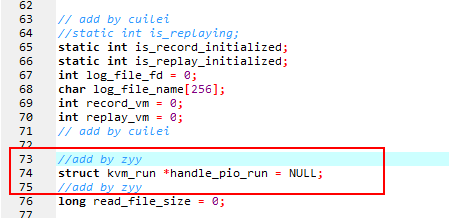
假设现在每次都只读一个扇区的数据，后期要是实验数据有变再修改

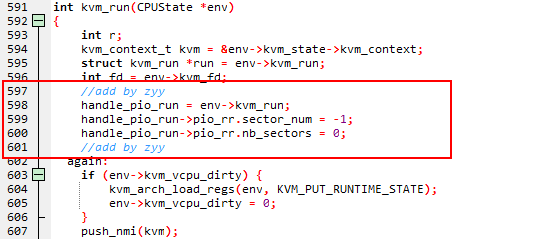
记录重放机器上/home/zhangyy/qrow/qemu-kvm/copy文件下保存的是经过下述修改的代码

现在进行相关测试，实现上述改进

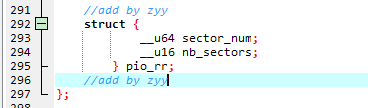
Qemu：

Qemu-kvm.c

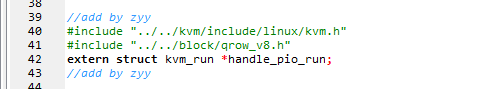


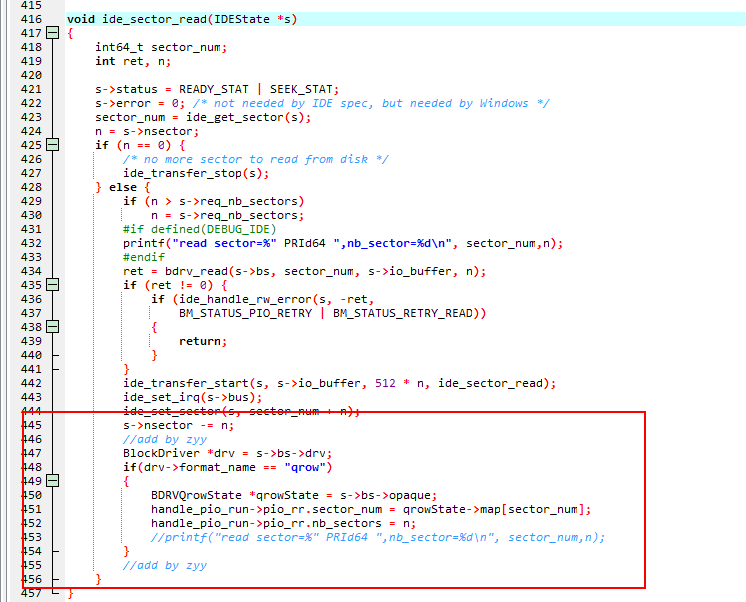


Kvm/include/linux/kvm.h



hw/ide/core.c

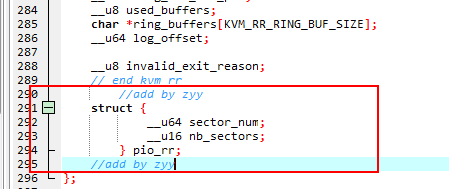




也可以在ide\_data\_readw里面进行上图类似操作，但是这样的话，会重复执行256遍，会不会影响效率

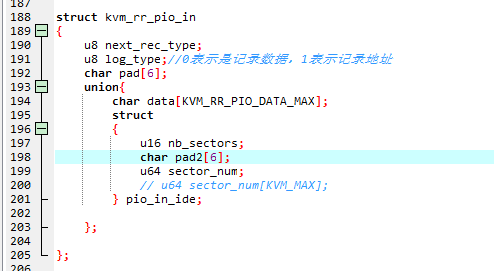
Kvm:

Include/linux/kvm.h



include/asm/kvm\_rr.h

include/asm-x86/kvm\_rr.h



\*/X86.c

//add by zyy

int kvm\_rr\_pio\_handle(struct kvm\_vcpu \*vcpu, struct kvm\_run \*kvm\_run)

{

//pio\_rr\_log.count = vcpu->arch.pio.count;

//pio\_rr\_log.port = vcpu->arch.pio.port;

//pio\_rr\_log.size = vcpu->arch.pio.size;

if(vcpu->is\_recording)

{

if((vcpu->arch.pio.count \* vcpu->arch.pio.size) > KVM\_RR\_PIO\_DATA\_MAX)

{

kvm\_debug("error...\n");

vcpu->is\_recording = 0;

return 0;

}

if(vcpu->log\_offset == -1 && vcpu->is\_recording)

{

struct kvm\_rr\_hdr hdr\_log;

hdr\_log.next\_rec\_type = 0;

// recording just started , write file header first

write\_log(KVM\_RR\_HEADER, vcpu, sizeof(struct kvm\_rr\_hdr), &hdr\_log);

}

struct kvm\_rr\_pio\_in pio\_rr\_log;

pio\_rr\_log.next\_rec\_type = 0;

if(vcpu->run->pio\_rr.sector\_num >= 0 && vcpu->run->pio\_rr.nb\_sectors >= 1 && (vcpu->arch.pio.count \* vcpu->arch.pio.size == 512))

{

pio\_rr\_log.log\_type = LOG\_TYPE\_ADDR;

pio\_rr\_log.pio\_in\_ide.sector\_num = vcpu->run->pio\_rr.sector\_num;

pio\_rr\_log.pio\_in\_ide.nb\_sectors = vcpu->run->pio\_rr.nb\_sectors;

//zyy

vcpu->run->pio\_rr.sector\_num = -1;

vcpu->run->pio\_rr.nb\_sectors = 0;

//zyy

write\_log(KVM\_RR\_PIO\_IN,vcpu,\

(sizeof(pio\_rr\_log.pio\_in\_ide) + 8), \

(void \*)&pio\_rr\_log);

}

else

{

pio\_rr\_log.log\_type = LOG\_TYPE\_DATA;

memcpy(pio\_rr\_log.data,vcpu->arch.pio\_data, \

vcpu->arch.pio.count \* vcpu->arch.pio.size);

kvm\_debug\_log("LOG\_IOIN %lu:%llu,%llx,%llx:port %d data %x count %d %d",\

vcpu->num\_recs,vcpu->rr\_ts.br\_count,vcpu->rr\_ts.rcx,vcpu->rr\_ts.rip,vcpu->arch.pio.port,\

\*(int \*)(vcpu->arch.pio\_data),vcpu->arch.pio.count,sizeof(struct kvm\_rr\_pio\_in));

write\_log(KVM\_RR\_PIO\_IN,vcpu,\

(vcpu->arch.pio.count \* vcpu->arch.pio.size + 8), \

(void \*)&pio\_rr\_log);

}

// reset counter to zero .. next event is relative

// from here

vcpu->rr\_ts.br\_count = 0;

// record pending pkts

kvm\_rr\_rec\_reqs(vcpu);

}// end of recording

else if(vcpu->is\_replaying)

{

struct kvm\_rr\_pio\_in \*pio\_rr\_log = NULL;

int ret;

ret = read\_log(vcpu);

//kvm\_err("%d \n", ret);

if(ret <= 0 || ret != KVM\_RR\_PIO\_IN)

{

// disable replaying , undefined behavior

kvm\_err("is out of sync %d expecting KVM\_RR\_PIO\_IN,\

got %d\n", ret != KVM\_RR\_PIO\_IN, ret);

vcpu\_disable\_rply(vcpu);

return 0;

}

else

{

// just copy the input data from log file

pio\_rr\_log = get\_log\_data\_ptr(vcpu);

}

if(!pio\_rr\_log)

{

// disable replaying , undefined behavior

kvm\_err("couldn't get data ptr\n");

vcpu\_disable\_rply(vcpu);

return 0;

}

else

{

vcpu->next\_rec\_type = pio\_rr\_log->next\_rec\_type;

u8 log\_type = pio\_rr\_log->log\_type;

if(log\_type == LOG\_TYPE\_DATA)

{

// copy to the place where user space would have

// copied

memcpy(vcpu->arch.pio\_data, pio\_rr\_log->data, \

vcpu->arch.pio.count \* vcpu->arch.pio.size);

kvm\_debug\_log("RPLY\_PIO %lu:%llu,%llx,%llx:port %d data %x count %d",\

vcpu->num\_recs,vcpu->rr\_ts.br\_count,vcpu->rr\_ts.rcx,vcpu->rr\_ts.rip,vcpu->arch.pio.port,\

\*(int \*)(vcpu->arch.pio\_data),vcpu->arch.pio.count);

vcpu->rr\_ts.br\_count = 0;

kvm\_rr\_rply\_reqs(vcpu);

return 1;

}

else

{

vcpu->run->pio\_rr.sector\_num = pio\_rr\_log->pio\_in\_ide.sector\_num;

vcpu->run->pio\_rr.nb\_sectors = pio\_rr\_log->pio\_in\_ide.nb\_sectors;

//接下来需要退出到qemu层读取数据。。。。

/\* 下面两行代码也不造要不要，怎样处理了

vcpu->rr\_ts.br\_count = 0;

kvm\_rr\_rply\_reqs(vcpu);

\*/

return -1;

}

}

}// end of replay

return 1;//没有记录重放

}

static int emulator\_pio\_in\_emulated(int size, unsigned short port, void \*val,

unsigned int count, struct kvm\_vcpu \*vcpu)

{

// kvm rr

struct kvm \*kvm\_run;

// data coming from user space

// end kvm rr

if (vcpu->arch.pio.count)

goto data\_avail;

trace\_kvm\_pio(0, port, size, count);

//回放的时候，是直接跳到了data avail执行么?不需要再跳到qemu处理了吧，后期需要验证一下

vcpu->arch.pio.port = port;

vcpu->arch.pio.in = 1;

vcpu->arch.pio.count = count;

vcpu->arch.pio.size = size;

if (!kernel\_pio(vcpu, vcpu->arch.pio\_data))

{

data\_avail:

// kvm rr

// record the data which is presented by kernel instead of

// user space

kvm\_run = vcpu->run;

int ret = kvm\_rr\_pio\_handle(vcpu,kvm\_run);

if (ret == 1)

{

memcpy(val, vcpu->arch.pio\_data, size \* count);

vcpu->arch.pio.count = 0;

return 1;

}

else

{

if (ret == -1)

{

//要去qemu拿数据了。。。

}

else

{

}

//证明 kvm\_rr\_pio\_handle()出错了。。。。没看到原代码做出任何应对措施

}

// end kvm rr

}

vcpu->run->exit\_reason = KVM\_EXIT\_IO;

vcpu->run->io.direction = KVM\_EXIT\_IO\_IN;

vcpu->run->io.size = size;

vcpu->run->io.data\_offset = KVM\_PIO\_PAGE\_OFFSET \* PAGE\_SIZE;

vcpu->run->io.count = count;

vcpu->run->io.port = port;

return 0;

}

//add by zyy

问题1：

安装同一个centos的操作系统

qcow2格式安装完后，镜像大小为705M

raw格式安装完后，镜像大小为695M

qrow 格式安装完后，镜像大小为1376 M 比前两种格式大太多太多了。。。。。

执行下面的命令后，镜像大小增加了2M

./x86\_64-softmmu/qemu-system-x86\_64 -hda /home/zhangyy/qrow/vm/centos-qrow-2.img -vnc :2

问题2：

记录pio\_in ide对应的512字节的扇区号而不是数据的话，回放的时候需要去qemu层处理，相对比较麻烦。。。。

问题3：

Qrow现在还是太慢了，安装系统，启动虚拟机等，都需要太久太久时间了