

Linux多队列网卡

多队列网卡是一种技术，最初是用来解决网络IO QoS（quality of service）问题的，后来随着网络IO的带宽的不断提升，单核CPU不能完全满足网卡的需求，通过多队列网卡驱动的支持，将各个队列通过中断绑定到不同的核上，以满足网卡的需求。

常见的有Intel的82575、82576，Boardcom的57711等，下面以公司的服务器使用较多的Intel 82575网卡为例，分析一下多队列网卡的硬件的实现以及linux内核软件的支持。

1.多队列网卡硬件实现

图1.1是Intel 82575硬件逻辑图，有四个硬件队列。当收到报文时，通过hash包头的SIP、Sport、DIP、Dport四元组，将一条流总是收到相同的队列。同时触发与该队列绑定的中断。

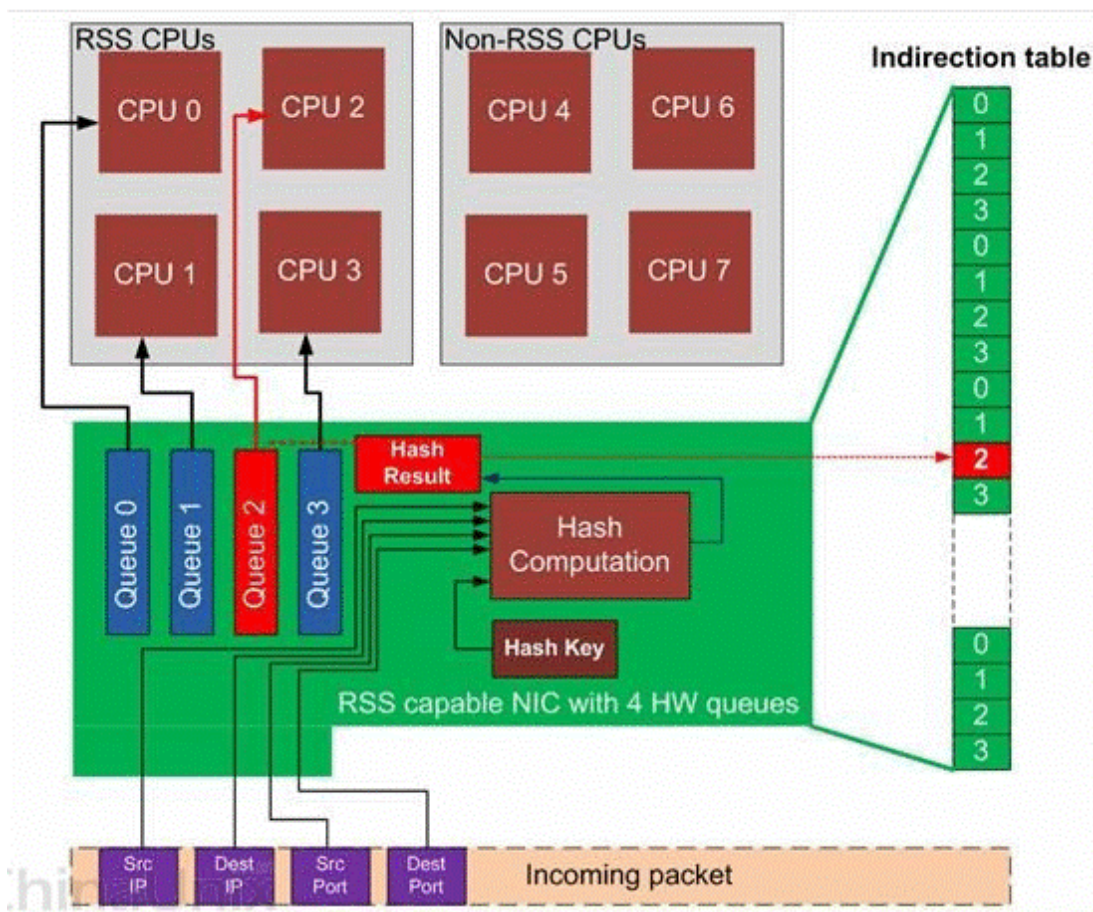


图1.1 82575硬件逻辑图

2. 2.6.21以前网卡驱动实现

kernel从2.6.21之前不支持多队列特性，一个网卡只能申请一个中断号，因此同一个时刻只有一个核在处理网卡收到的包。如图2.1，协议栈通过NAPI轮询收取各个硬件queue中的报文到图2.2的net_device数据结构中，通过QDisc队列将报文发送到网卡。

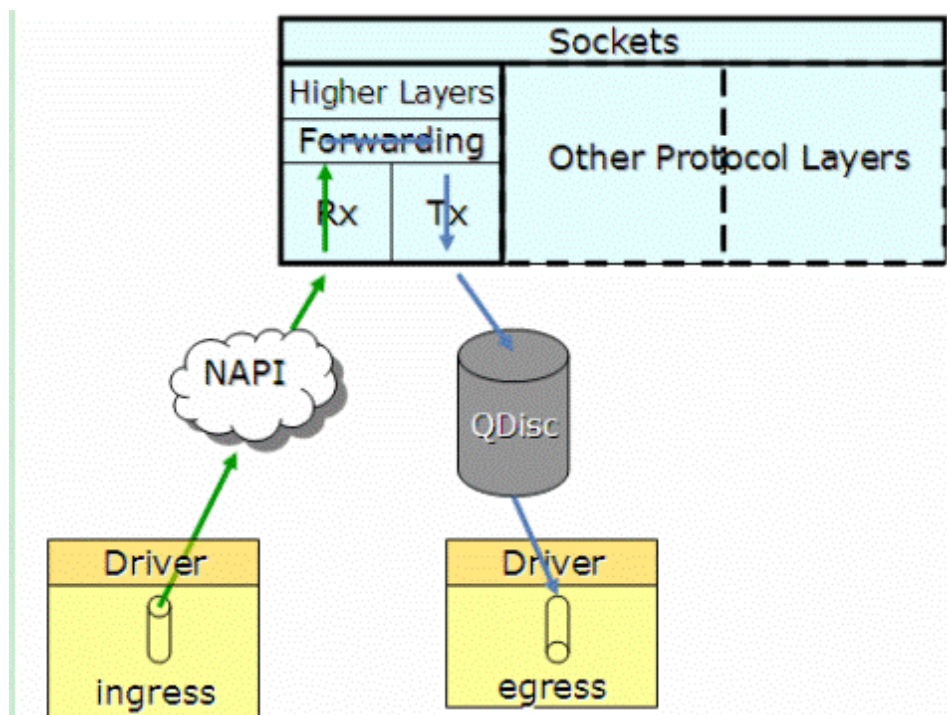


图2.1 2.6.21之前内核协议栈

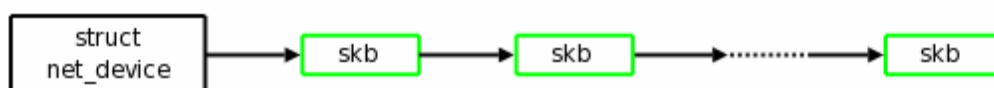


图2.2 2.6.21之前net_device

3. 2.6.21后网卡驱动实现

2.6.21开始支持多队列特性，当网卡驱动加载时，通过获取的网卡型号，得到网卡的硬件queue的数量，并结合CPU核的数量，最终通过 $Sum = \min(\text{网卡queue}, \text{CPU core})$ 得出所要激活的网卡queue数量（Sum），并申请Sum个中断号，分配给激活的各个queue。

如图3.1，当某个queue收到报文时，触发相应的中断，收到中断的核，将该任务加入到协议栈负责收包的该核的NET_RX_SOFTIRQ队列中（NET_RX_SOFTIRQ在每个核上都有一个实例），在NET_RX_SOFTIRQ中，调用NAPI的收包接口，将报文收到CPU中如图3.2的有多个netdev_queue的net_device数据结构中。

这样，CPU的各个核可以并发的收包，就不会应为一个核不能满足需求，导致网络IO性能下降。

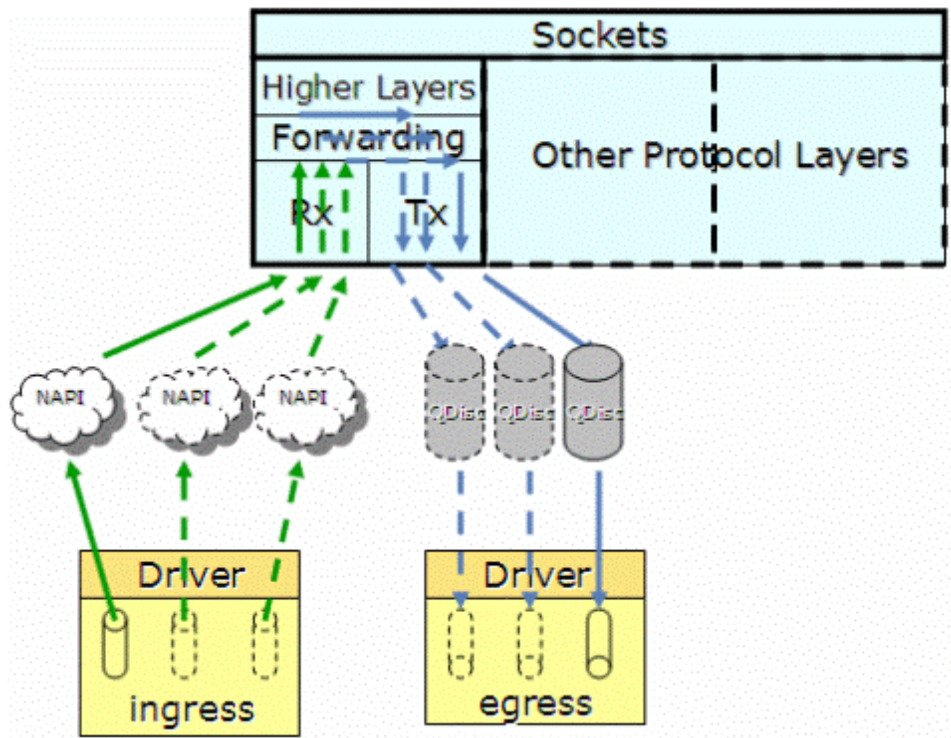


图3.1 2.6.21之后内核协议栈

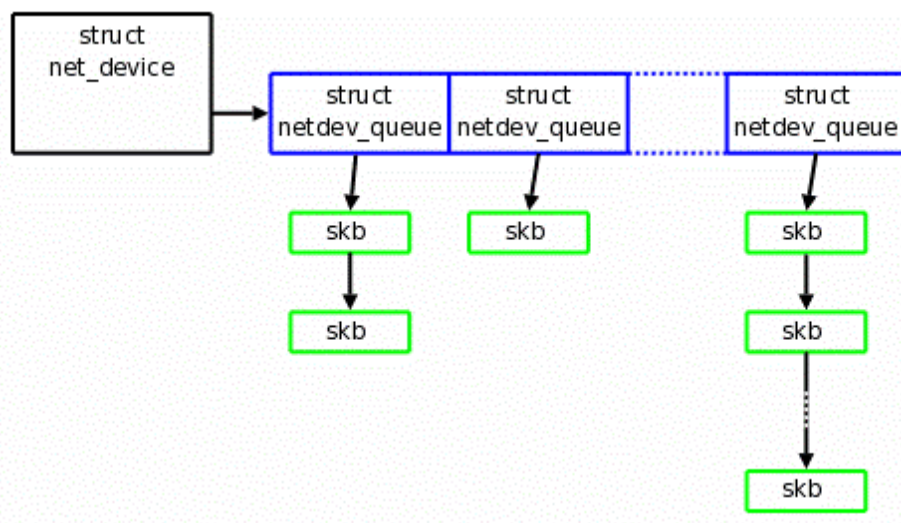


图3.2 2.6.21之后net_device

4.中断绑定

当CPU可以平行收包时，就会出现不同的核收取了同一个queue的报文，这就会产生报文乱序的问题，解决方法是将一个queue的中断绑定到唯一的一个核上去，从而避免了乱序问题。同时如果网络流量大的时候，可以将软中断均匀分散到各个核上，避免CPU成为瓶颈。

52:	0	0	0	0	PCI-MSI-edge	eth0
53:	294242	0	0	0	PCI-MSI-edge	eth0-rx-0
54:	22	0	0	321436	PCI-MSI-edge	eth0-rx-1
55:	311058	0	0	0	PCI-MSI-edge	eth0-rx-2
56:	22	0	326321	0	PCI-MSI-edge	eth0-rx-3
57:	352432	0	0	0	PCI-MSI-edge	eth0-tx-0
58:	320008	32	0	0	PCI-MSI-edge	eth0-tx-1
59:	25	298057	0	0	PCI-MSI-edge	eth0-tx-2
60:	392454	0	0	0	PCI-MSI-edge	eth0-tx-3

图4.1 /proc/interrupts

5.中断亲合纠正

一些多队列网卡驱动实现的不是太好，在初始化后会出现图4.1中同一个队列的tx、rx中断绑定到不同核上的问题，这样数据在core0与core1之间流动，导致核间数据交互加大，cache命中率降低，降低了效率。

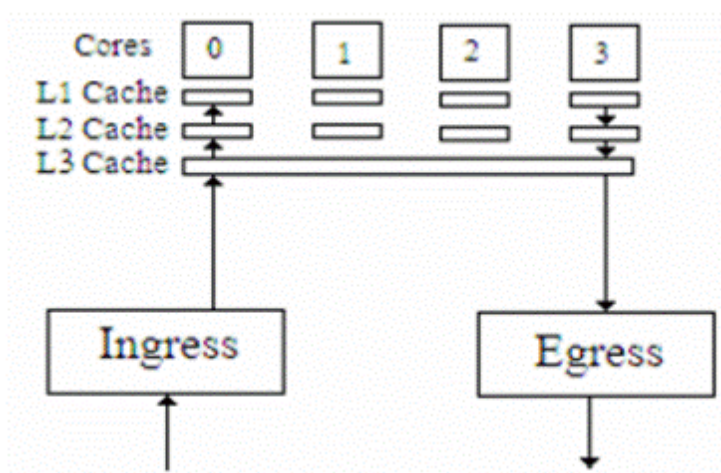


图5.1 不合理中断绑定

linux network子系统的负责人David Miller提供了一个脚本，首先检索/proc/interrupts文件中的信息，按照图4.1中eth0-rx-0 (\$VEC) 中的VEC得出中断MASK，并将MASK写入中断号53对应的smp_affinity中。由于eth-rx-0与eth-tx-0的VEC相同，实现同一个queue的tx与rx中断绑定到一个核上，如图4.3所示。

```
set_affinity()
{
    MASK=$((1<<$VEC))
    printf "%s mask=%X for /proc/irq/%d/smp_affinity\n" $DEV $MASK $IRQ
    printf "%X" $MASK > /proc/irq/$IRQ/smp_affinity
    #echo $DEV mask=$MASK for /proc/irq/$IRQ/smp_affinity
    #echo $MASK > /proc/irq/$IRQ/smp_affinity
}
```

```
eth0 mask=1 for /proc/irq/53/smp_affinity
eth0 mask=2 for /proc/irq/54/smp_affinity
eth0 mask=4 for /proc/irq/55/smp_affinity
eth0 mask=8 for /proc/irq/56/smp_affinity
eth0 mask=1 for /proc/irq/57/smp_affinity
eth0 mask=2 for /proc/irq/58/smp_affinity
eth0 mask=4 for /proc/irq/59/smp_affinity
eth0 mask=8 for /proc/irq/60/smp_affinity
```

图4.2 set_irq_affinity

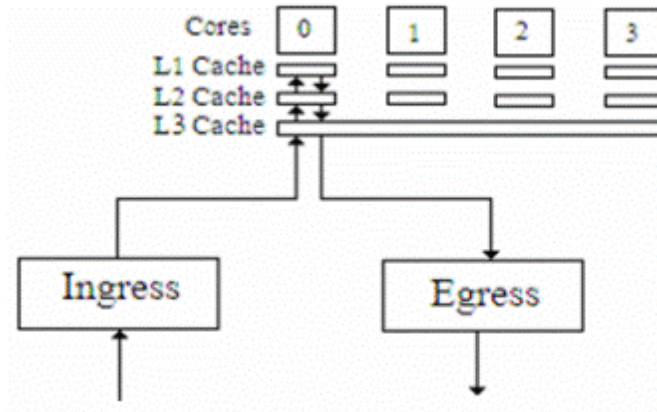


图4.3 合理的中断绑定

set_irq_affinity脚本位于http://mirror.oa.com/tlinux/tools/set_irq_affinity.sh。

6.多队列网卡识别

#lspci -vvv

Ethernet controller的条目内容，如果有MSI-X && Enable+ && TabSize > 1，则该网卡是多队列网卡，如图4.4所示。

```
04:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection
Subsystem: Inventec Corporation Device 004b
Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr+ Ste
Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort-
Latency: 0, Cache Line Size: 64 bytes
Interrupt: pin A routed to IRQ 16
Region 0: Memory at faf60000 (32-bit, non-prefetchable) [size=128K]
Region 1: Memory at faf40000 (32-bit, non-prefetchable) [size=128K]
Region 2: I/O ports at d880 [size=32]
Region 3: Memory at fafb8000 (32-bit, non-prefetchable) [size=16K]
Expansion ROM at faf20000 [disabled] [size=128K]
Capabilities: [40] Power Management version 3
        Flags: PMEClk- DSI+ D1- D2- AuxCurrent=0mA PME(D0+,D1-,D2-,D3ho
Status: D0 PME-Enable- DSel=0 DScale=1 PME-
Capabilities: [50] Message Signalled Interrupts: Mask+ 64bit+ Count=1/1
        Address: 00000000 Data: 0000
        Masking: 00000000 Pending: 00000000
Capabilities: [70] MSI-X: Enable+ Mask- TabSize=10
```

图4.4 lspci内容

Message Signaled Interrupts(MSI)是PCI规范的一个实现，可以突破CPU 256条interrupt的限制，使每个设备具有多个中断线变成可能，多队列网卡驱动给每个queue申请了MSI。

MSI-X是MSI数组，Enable+指使能，TabSize是数组大小。

```

# setting up irq affinity according to /proc/interrupts
# 2008-11-25 Robert Olsson
# 2009-02-19 updated by Jesse Brandeburg
#
# > Dave Miller:
# (To get consistent naming in /proc/interrupts)
# I would suggest that people use something like:
# char buf[IFNAMSIZ+6];
#
# sprintf(buf, "%s-%s-%d",
#     netdev->name,
#     (RX_INTERRUPT ? "rx" : "tx"),
#     queue->index);
#
# Assuming a device with two RX and TX queues.
# This script will assign:
#
# eth0-rx-0 CPU0
# eth0-rx-1 CPU1
# eth0-tx-0 CPU0
# eth0-tx-1 CPU1
#
set_affinity()
{
    MASK=$((1<<$VEC))
    printf "%s mask=%X for /proc/irq/%d/smp_affinity\n" $DEV $MASK $IRQ
    printf "%X" $MASK > /proc/irq/$IRQ/smp_affinity
    #echo $DEV mask=$MASK for /proc/irq/$IRQ/smp_affinity
    #echo $MASK > /proc/irq/$IRQ/smp_affinity
}
if [ "$1" = "" ] ; then

```

```

echo "Description:"
echo "  This script attempts to bind each queue of a multi-queue NIC"
echo "  to the same numbered core, ie tx0|rx0 --> cpu0, tx1|rx1 --> cpu1"
echo "usage:"
echo "  $0 eth0 [eth1 eth2 eth3]"
fi

# check for irqbalance running
IRQBALANCE_ON=`ps ax | grep -v grep | grep -q irqbalance; echo $?`
if [ "$IRQBALANCE_ON" == "0" ] ; then
echo " WARNING: irqbalance is running and will"
echo "      likely override this script's affinitization."
echo "      Please stop the irqbalance service and/or execute"
echo "      'killall irqbalance'"
fi

#
# Set up the desired devices.
#
for DEV in $*
do
  for DIR in rx tx TxRx
  do
    MAX=`grep $DEV-$DIR /proc/interrupts | wc -l`
    if [ "$MAX" == "0" ] ; then
      MAX=`egrep -i "$DEV:.*$DIR" /proc/interrupts | wc -l`
    fi
    if [ "$MAX" == "0" ] ; then
      echo no $DIR vectors found on $DEV
      continue
    fi
    #exit 1
  fi
  for VEC in `seq 0 1 $MAX`
  do
    IRQ=`cat /proc/interrupts | grep -i $DEV-$DIR-$VEC"$" | cut -d: -f1 | sed "s/ //g"`

```

```

if [ -n "$IRQ" ]; then
    set_affinity
else
    IRQ=`cat /proc/interrupts | egrep -i $DEV:v$VEC-$DIR"$" | cut -d: -f1 | sed "s/ //g"`
    if [ -n "$IRQ" ]; then
        set_affinity
    fi
fi
done
done
done

```

--	--	--

多队列网卡CPU中断均衡

一、基础

1.相关名词

IRQ

Interrupt Request,中断请求，从硬件层发出

作用：执行硬件中断的请求

SMP (Symmetrical Multi-Processing)

对称多处理器系统，是指在一个计算机上汇集了一组CPU，各CPU之间共享内存子系统以及总线结构（或者说是两个或多个同样的处理器通过一块共享内存彼此连接。）

作用：适用于多处理器计算机

APIC(Advanced Programmable Interrupt Controllers)

高级可编程中断控制器

松耦合多处理架构

最早的Linux SMP是松耦合多处理系统。这些系统是利用多个高速互连的单一系统构造的（如 10G 以太网、Fibre Channel 或 Infiniband）。构建松耦合多处理系统很容易，但是构建大型的多处理器网络可能占用相当大的空

间并消耗很多电量。因为它们通常是利用普通硬件来构建的，所以包含的有些硬件不相关却要耗费很多电量和空间。更大的缺点在于通信结构。即使使用高速网络（如 10G 以太网），也存在系统可伸缩性的限制。

CMP

芯片多级处理。CMP一种紧密耦合多处理器，可以将它看作将松耦合架构缩小至芯片级。即在一个集成电路中，多个芯片、共享内存以及互连形成了一个紧密集成的多处理核心

2.中断的相关概念

Linux 内核对计算机上所有的设备进行管理，进行管理的方式是内核和设备之间的通信。解决通信的方式有两种：

1. 轮询。轮询是指内核对设备状态进行周期性的查询
2. 中断。中断是指在设备需要CPU的时候主动发起通信

从物理学的角度看，中断是一种电信号，由硬件设备产生，并直接送入中断控制器（如 8259A）的输入引脚上，然后再由中断控制器向处理器发送相应的信号。处理器一经检测到该信号，便中断自己当前正在处理的工作，转而去处理中断。此后，处理器会通知 OS 已经产生中断。这样，OS 就可以对这个中断进行适当的处理。不同的设备对应的中断不同，而每个中断都通过一个唯一的数字标识，这些值通常被称为中断线。

- **中断可以分为NMI（不可屏蔽中断）和INTR（可屏蔽中断）。**

其中 NMI 是不可屏蔽中断，它通常用于电源掉电和物理存储器奇偶校验；INTR是可屏蔽中断，可以通过设置中断屏蔽位来进行中断屏蔽，它主要用于接受外部硬件的中断信号，这些信号由中断控制器传递给 CPU。

- **常见的两种中断控制器：**

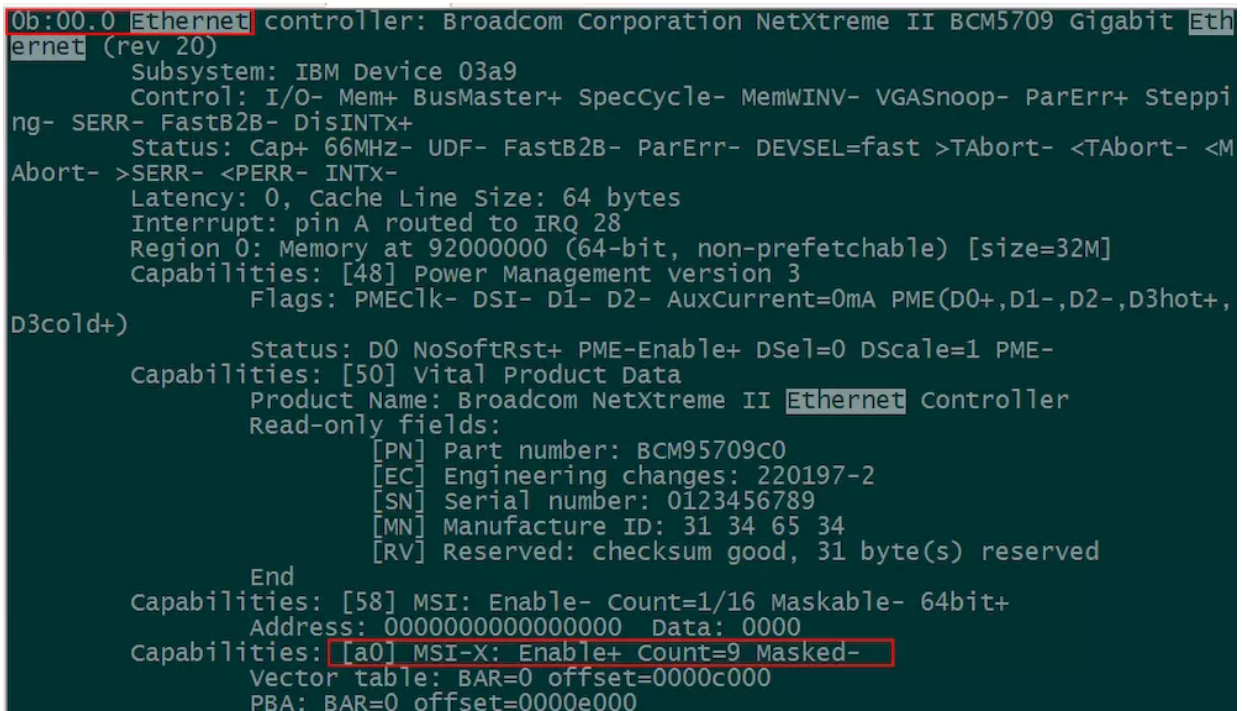
- 1.可编程中断控制器（PIC）8259A
- 2.高级可编程中断控制器（APIC）

传统的 PIC（Programmable Interrupt Controller）是由两片 8259A 风格的外部芯片以“级联”的方式连接在一起。每个芯片可处理多达 8 个不同的 IRQ。因为从 PIC 的 INT 输出线连接到主 PIC 的 IRQ2 引脚，所以可用 IRQ 线的个数达到 15 个

二、多队列网卡CPU中断均衡

注意：本文全部是基于多核CPU环境而写，如果是单核CPU，没有任何意义
首先。我们要先判断当前系统环境是否支持多队列网卡，执行命令：

lspci -vvv



```
0b:00.0 Ethernet controller: Broadcom Corporation NetXtreme II BCM5709 Gigabit Ethernet (rev 20)
Subsystem: IBM Device 03a9
Control: I/O- Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr+ Stepping- SERR- FastB2B- DisINTx+
Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR- INTx-
Latency: 0, Cache Line Size: 64 bytes
Interrupt: pin A routed to IRQ 28
Region 0: Memory at 92000000 (64-bit, non-prefetchable) [size=32M]
Capabilities: [48] Power Management version 3
Flags: PMEClk- DSI- D1- D2- AuxCurrent=0mA PME(D0+,D1-,D2-,D3hot+,D3cold+)
Status: D0 NoSoftRst+ PME-Enable+ DSel=0 DScale=1 PME-
Capabilities: [50] Vital Product Data
Product Name: Broadcom NetXtreme II Ethernet Controller
Read-only fields:
[PN] Part number: BCM95709C0
[EC] Engineering changes: 220197-2
[SN] Serial number: 0123456789
[MN] Manufacture ID: 31 34 65 34
[RV] Reserved: checksum good, 31 byte(s) reserved
End
Capabilities: [58] MSI: Enable- Count=1/16 Maskable- 64bit+
Address: 0000000000000000 Data: 0000
Capabilities: [a0] MSI-X: Enable+ Count=9 Masked-
Vector table: BAR=0 offset=0000c000
PBA: BAR=0 offset=0000e000
```

Paste_Image.png

注意上图中红色部分。如果在Ethernet项中。含有**[a0] MSI-X: Enable+ Count=9 Masked-**语句，则说明当前系统环境是支持多队列网卡的，否则不支持。对于不支持多网卡队列的CPU均衡，将在下文分析。

1. Linux系统中断设置原理

为了在支持SMP的硬件上通过Linux使用SMP，需要适当的配置内核。

以网卡中断为例，如果流量大了，一个CPU处理会十分吃力，甚至崩溃；因为CPU处于忙碌状态，此时系统性能低，不能较快处理网卡接收的数据包。数据包堆积，网卡缓存溢出，导致丢包。这也是为什么负载高时会有网络丢包的原因了。但是服务器明明是多核CPU，为什么其他CPU没有参与处理了？原因是CPU没有做均衡时，默认是在CPU0上执行中断。虽然系统服务本身是有/etc/init.d/irqbalance 这个服务的作用就是用来做CPU均衡的，但是对于处理流量很大的服务器来说，这个服务的效果就微乎其微了，CPU均衡没有达到最优。

注意：如果要使用SMP（对称多处理系统），CPU需要内置APIC

中断绑定（CPU均衡）分为单队列网卡和多队列网卡两种情况。对于多队列网卡，开启SMP，如果只是开启SMP可能不会使CPU中断均衡达到最优。

可以同时开启SMP和RPS/RFS，使得CPU中断均衡达到最优（因为CPU核心可能会更多，但是网卡队列只有4-8个之类的，这个需要看具体机机型）。对于单队列网，只能开启RPS/RFS。

中断绑定——中断亲和力 (IRQ Affinity)

维持亲和性是为了提高缓存效率

在 SMP 体系结构中，我们可以通过调用系统调用和一组相关的宏来设置 CPU 亲和力 (CPU affinity)，将一个或多个进程绑定到一个或多个处理器上运行。中断在这方面也毫不示弱，也具有相同的特性——中断亲和力。中断亲和力是指将一个或多个中断源绑定到特定的 CPU 核心上运行。

在 /proc/irq 目录中，对于已经注册中断处理程序的硬件设备，都会在该目录下存在一个以该中断号命名的目录，该目录下有一个 **smp_affinity 文件 (SMP 体系结构才有该文件)**，文件中的数据表示 **CPU 位掩码**，可以用来设置该IRQ与某个CPU的亲和力（默认值为 0xffffffff，表明把中断发送到所有的 CPU 上去处理），通过指定CPU 核心与某个中断的亲和性后，中断所对应的硬件设备发出的中断请求就都会给这个CPU核心处理。

如果中断控制器不支持 IRQ affinity,不能改变此默认值，同时也不能关闭所有的 CPU 位掩码，即不能设置成 0x0。

注意：SMP 绑定irq 到网卡 只对多队列网卡生效。其实也可以通过查看/proc/interrupt来查看系统是否支持多队列网卡——即，如果interrupt文件中含有ethN-xxx的就是多队列，如果只是ethN的就是单队列

**

2.相关目录文件以及实例

2.1中断相关文件

- **/proc/interrupts**: 该文件存放了每个I/O设备的对应中断号、每个CPU的中断数、中断类型。
- **/proc/irq/**: 该目录下存放的是以IRQ号命名的目录，如/proc/irq/40/，表示中断号为40的相关信息
- **/proc/irq/[irq_num]/smp_affinity**: 该文件存放的是CPU位掩码（十六进制）。修改该文件中的值可以改变CPU和某中断的亲和性
- **/proc/irq/[irq_num]/smp_affinity_list**: 该文件存放的是CPU列表（十进制）。注意，CPU核心个数用表示编号从0开始，如cpu0,cpu1等

- smp_affinity_list和smp_affinity任意更改一个文件都会生效，两个文件相互影响，只不过是表示方法不一致，但一般都是修改smp_affinity 文件

- 以8核CUP为例，列出相关文件中如何表示CPU列表

cup	二进制	smp_affinity_list	smp_affinity (十六进制)
cpu0	0001	0	1
cpu1	0010	1	2
cpu2	0100	2	4
cpu3	1000	3	8
cpu4	010000	4	10
cpu5	0100000	5	40
.....	如上类推		

在计算cpu亲和性时很容易混淆，我们先排除smp_affinity_list这项不看。只看二进制和十六进制这两项。这里可以得出一个公式：

Python语法：

```
smp_affinity_value = hex(2**(N-1))
```

其中N代表的是CPU核心数。那么，为什么是"N-1"呢。原因很简单，因为对于多核服务器而言，cpu编号是从cpu0开始的。比如24核心的服务器，cpu编号为cpu0-cpu23。

2.2 中断亲和性设置实例——以以太网卡的中断为例

- 动态监控CPU中断情况，观察中断变化

```
watch -d -n 1 'cat /proc/interrupts'
```

- 查看网卡中断相关信息

```
cat /proc/interrupts | grep -E "eth|CPU"
```

- 网卡亲和性设置

修改proc/irq/irq_number/smp_affinity之前，先停掉irq自动调节服务，不然修改的值就会被覆盖。

```
/etc/init.d/irqbalance stop
```

通过查看网卡中断相关信息，得到网卡中断为19

```
[root@master ~]# cd /proc/irq/19
[root@master 19]# cat smp_affinity
00000000,00000000,00000000,00000001
[root@master 19]# cat smp_affinity_list
0
```

修改值，将19号中断绑定在cpu2上：

```
[root@master 19]# echo 4 > smp_affinity
[root@master 19]# cat smp_affinity
00000000,00000000,00000000,00000004
[root@master 19]# cat smp_affinity_list
2
```

如果是要将网卡中断绑定在cpu0和cpu2上怎么做了？请先参照上文中的**CPU列表**。cpu0和2的十六进制值分别为1,4。那么如果要同时绑定在cpu0和cpu2上，则十六进制值为5，如下：

```
[root@master 19]# echo 5 > smp_affinity
[root@master 19]# cat smp_affinity
00000000,00000000,00000000,00000005
[root@master 19]# cat smp_affinity_list
0,2
```

再做一个连续绑定的例子，如绑定在cpu0,1,2上：

```
[root@master 19]# cat smp_affinity
00000000,00000000,00000000,00000007
[root@master 19]# cat smp_affinity_list
0-2
```

从这里可以得出一个结论：绑定单个cpu只要写数字就行，如果是绑定多个cpu则用逗号隔开，如果是绑定连续CPU，则用-符号。

注意：写入smp_affinity中的必须是16进制（不带0x标识），更新这个文件后，smp_affinity_list也会更新，这个文件里面是10进制。

我们在计算CPU 核心编号时，是以二进制算的，但是文件中要存放的是十六进制（不带0x标识）。如CPU 0表示的是第一个核心，二进制为0001，十六进制为1。该算法可参考前述的CPU 列表对应关系。

再比如，f 十六进制是15,二进制就是1111，这个就是表示设备随机选择一个CPU执行中断。

三、使用taskset为系统进程PID设置CPU亲和性

- 查看某个进程的CPU亲和性

```
# taskset -p 30011
pid 30011's current affinity mask: ff
```

- 设置某个进程的CPU亲和性

```
# taskset -p 1 30011
pid 30011's current affinity mask: ff
pid 30011's new affinity mask: 1
```

- 使用-c选项可以将一个进程对应到多个CPU上去

```
# taskset -p -c 1,3 30011
pid 30011's current affinity list: 0
```



```
pid 30011's new affinity list: 1,3
```

```
# taskset -p -c 1-7 30011
```

```
pid 30011's current affinity list: 1,3
```

```
pid 30011's new affinity list: 1-7
```

四、多队列网卡中断绑定——CPU中断均衡脚本

eth_irq.py

```
#!/usr/bin/python
```

```
import re
from multiprocessing import cpu_count

dir = '/proc/irq'
interrupt = '/proc/interrupts'

class IRQ():
    def __init__(self):
        self.irq_num = []

        with open(interrupt, 'r') as f:
            for i in f:
                if re.search(r'eth|em', i):
                    #print re.split(r'\s*|:', i)[1]
                    self.irq_num.append(re.split(r'\s*|:', i)
[1].split(':')[0])
                    #if re.search('eth', i):
                    #    print re.split(r'\s*|:', i)
                    #    self.irq_num.append(re.split(r'\s*|:', i)[2])
        print self.irq_num
        self.cpu_num = cpu_count()
        self.mask = [hex(2**i).split('0x')[1] for i in
range(self.cpu_num)]
        self.set_affinity()

    def set_affinity(self):
        affinity_file = []
        for i in self.irq_num:
            affinity_file.append('%s/%s/smp_affinity' % (dir, i))
        #print affinity_file
        #print self.mask
        self.mask.extend(self.mask)
        #print self.mask
```

```

        for i in range(len(self.mask)):
            print '%s %s' % (self.mask[i], affinity_file[i])
            with open(affinity_file[i], 'w') as f:
                f.write("%s" % self.mask[i])

if __name__ == '__main__':
    a = IRQ()
    print a.mask

```

这个脚本写的有点死，不够灵活，有时间再重写一下，做一个封装。

执行上面脚本运行前，可执行test.sh查看smp_affinity_list中的值的变化

test.sh

```

#!/bin/bash

irq=`grep 'eth' /proc/interrupts | awk '{print $1}' | cut -d : -f 1`
for i in $irq
do
    num=`cat /proc/irq/$i/smp_affinity_list`
    echo /proc/irq/$i/smp_affinity_list "$num"
done

```

前（其实之前已经做过均衡，我这里只是改了下）

```

/proc/irq/59/smp_affinity_list    2
/proc/irq/60/smp_affinity_list    3
/proc/irq/61/smp_affinity_list    4
/proc/irq/62/smp_affinity_list    5
/proc/irq/63/smp_affinity_list    6
/proc/irq/64/smp_affinity_list    7
/proc/irq/65/smp_affinity_list    0
/proc/irq/66/smp_affinity_list    1
/proc/irq/68/smp_affinity_list    2
/proc/irq/69/smp_affinity_list    3
/proc/irq/70/smp_affinity_list    4
/proc/irq/71/smp_affinity_list    5
/proc/irq/72/smp_affinity_list    6
/proc/irq/73/smp_affinity_list    7
/proc/irq/74/smp_affinity_list    0
/proc/irq/75/smp_affinity_list    1

```

后

```

/proc/irq/59/smp_affinity_list    0
/proc/irq/60/smp_affinity_list    1
/proc/irq/61/smp_affinity_list    2
/proc/irq/62/smp_affinity_list    3
/proc/irq/63/smp_affinity_list    4

```

```

/proc/irq/64/smp_affinity_list    5
/proc/irq/65/smp_affinity_list    6
/proc/irq/66/smp_affinity_list    7
/proc/irq/68/smp_affinity_list    0
/proc/irq/69/smp_affinity_list    1
/proc/irq/70/smp_affinity_list    2
/proc/irq/71/smp_affinity_list    3
/proc/irq/72/smp_affinity_list    4
/proc/irq/73/smp_affinity_list    5
/proc/irq/74/smp_affinity_list    6
/proc/irq/75/smp_affinity_list    7

```

五、单队列多网卡CPU中断均衡

使用RPS/RFS在软件层面模拟多队列网卡功能。RPS/RFS是谷歌工程师提交的内核补丁。意在处理多核CPU单队列网卡的情况。

RFS需要内核编译CONFIG_RPS选项，RFS才起作用。全局数据流表(rps_sock_flow_table)的总数可以通过下面的参数来设置：

```
/proc/sys/net/core/rps_sock_flow_entries
```

每个队列的数据流表总数可以通过下面的参数来设置：

```

/sys/class/net/[iface]/queues/rx-/rps_cpus
/sys/class/net/[iface]/queues/rx-/rps_flow_cnt
/proc/sys/net/core/rps_sock_flow_entries

```

- /sys/class/net/[iface]/queues/rx-/rps_cpus

该文件存放的是对应的CPU核心，如果值为f...则表示每个队列绑定到所有cpu核心上；如果值为1，2之类的，则表示为绑定对应的CPU核心。如：对于物理CPU个数为2，逻辑CPU为8核心的机器，具体计算方法是第一颗cpu是00000001，第二个cpu是00000010，第3个cpu是00000100，依次类推，由于是所有的cpu都负担，所以所有的cpu数值相加，得到的数值为11111111，十六进制就刚好是ff。ff就表示绑定到所有CPU核心上。

- /proc/sys/net/core/rps_sock_flow_entries

该数值是根据网卡有多少个个通道计算得出的数据，例如8通道的网卡，那么1个网卡，每个通道设置4096的数值，8*4096就是/proc/sys/net/core/rps_sock_flow_entries 的数值，对于内存大的机器可以适当调大rps_flow_cnt

每个队列分别绑定到一个对应的CPU核心上

```
/sys/class/net/eth1/queues/rx-0/rps_cpus 1 (这里的数据是十六进制，文件中为：000001)
/sys/class/net/eth1/queues/rx-1/rps_cpus 2
/sys/class/net/eth1/queues/rx-2/rps_cpus 4
/sys/class/net/eth1/queues/rx-3/rps_cpus 8
/sys/class/net/eth1/queues/rx-4/rps_cpus 10
/sys/class/net/eth1/queues/rx-5/rps_cpus 20
/sys/class/net/eth1/queues/rx-6/rps_cpus 40
/sys/class/net/eth1/queues/rx-7/rps_cpus 80

/sys/class/net/eth1/queues/rx-0/rps_flow_cnt 4096
/sys/class/net/eth1/queues/rx-1/rps_flow_cnt 4096
/sys/class/net/eth1/queues/rx-2/rps_flow_cnt 4096
/sys/class/net/eth1/queues/rx-3/rps_flow_cnt 4096
/sys/class/net/eth1/queues/rx-4/rps_flow_cnt 4096
/sys/class/net/eth1/queues/rx-5/rps_flow_cnt 4096
/sys/class/net/eth1/queues/rx-6/rps_flow_cnt 4096
/sys/class/net/eth1/queues/rx-7/rps_flow_cnt 4096

/proc/sys/net/core/rps_sock_flow_entries 32768
```

每个队列绑定到所有CPU核心上

```
/sys/class/net/eth1/queues/rx-0/rps_cpus ff
/sys/class/net/eth1/queues/rx-1/rps_cpus ff
/sys/class/net/eth1/queues/rx-2/rps_cpus ff
/sys/class/net/eth1/queues/rx-3/rps_cpus ff
/sys/class/net/eth1/queues/rx-4/rps_cpus ff
/sys/class/net/eth1/queues/rx-5/rps_cpus ff
/sys/class/net/eth1/queues/rx-6/rps_cpus ff
/sys/class/net/eth1/queues/rx-7/rps_cpus ff

/sys/class/net/eth1/queues/rx-0/rps_flow_cnt 4096
/sys/class/net/eth1/queues/rx-1/rps_flow_cnt 4096
/sys/class/net/eth1/queues/rx-2/rps_flow_cnt 4096
/sys/class/net/eth1/queues/rx-3/rps_flow_cnt 4096
/sys/class/net/eth1/queues/rx-4/rps_flow_cnt 4096
/sys/class/net/eth1/queues/rx-5/rps_flow_cnt 4096
/sys/class/net/eth1/queues/rx-6/rps_flow_cnt 4096
/sys/class/net/eth1/queues/rx-7/rps_flow_cnt 4096
```

```
/proc/sys/net/core/rps_sock_flow_entries 32768
```

如果不开启rps功能，则rps_cpus 文件中的值设置为0

七、IRQ均衡脚本

写脚本时发现很多问题，如下面的两台主机，都是支持多队列网卡的，但是注意观察最后两列的区别

```
[root@master ~]# grep 'eth' /proc/interrupts
```

```
59: 61033490 0 3671863 0 0 0
0 0 PCI-MSI-edge eth0-0
60: 41459456 3308875 0 2914342 0 0
0 0 PCI-MSI-edge eth0-1
61: 54622748 0 3360309 0 4103929 0
0 0 PCI-MSI-edge eth0-2
62: 310768180 0 0 32393316 0 31050595
0 0 PCI-MSI-edge eth0-3
63: 47763053 0 0 0 3674309 0
3352638 0 PCI-MSI-edge eth0-4
64: 66969322 0 0 0 0 5011122
0 5180864 PCI-MSI-edge eth0-5
65: 50396675 0 0 0 0 0
3175900 0 PCI-MSI-edge eth0-6
66: 44104243 3138376 0 0 0 0
0 3091676 PCI-MSI-edge eth0-7
68: 3994017501 0 338732695 0 0 0
0 0 PCI-MSI-edge eth1-0
69: 2203747223 766400094 0 818107598 0 0
0 0 PCI-MSI-edge eth1-1
70: 3089604544 0 759400051 0 795843526 0
0 0 PCI-MSI-edge eth1-2
71: 1894677558 0 0 760811049 0 793309576
0 0 PCI-MSI-edge eth1-3
72: 805024305 0 0 0 723044628 0
759874105 0 PCI-MSI-edge eth1-4
73: 1582319475 0 0 0 0 721360118
0 781055635 PCI-MSI-edge eth1-5
74: 2854078786 0 0 0 0 0
709514558 0 PCI-MSI-edge eth1-6
75: 1699542056 779165245 0 0 0 0
0 717908550 PCI-MSI-edge eth1-7
```

```
[root@minion ~]# grep 'eth' /proc/interrupts
```

```
90: 6 0 0 0 0 0
0 0 PCI-MSI-X eth0
98: 3178016492 23261814 26903212 24899595 2274387599 284769513
2180244656 385019532 PCI-MSI-X eth0-rx-0
106: 1442 411400813 25969662 6659594 2117294006 142635319
2079125696 331603756 PCI-MSI-X eth0-rx-1
114: 6626 20485044 2713744686 6831426 3331230251 4042587617
992902361 1019496640 PCI-MSI-X eth0-rx-2
122: 771 38602882 23569683 1670873952 1400480565 1479682722
878554708 1266571848 PCI-MSI-X eth0-rx-3
130: 676 4001308 121711 241193970 4054343807 10
2219979772 2992231001 PCI-MSI-X eth0-tx-0
146: 8 0 0 0 0 0
```


0	0	PCI-MSI-X	eth1		
154:	1208	304770485	506778587	3164891700	6749496 3992464600
7582534	610809	PCI-MSI-X	eth1-rx-0		
162:	24012	923562439	1868789464	430527766	5200172 4789746
978408581	297773	PCI-MSI-X	eth1-rx-1		
170:	179444	589446443	591027856	173978259	2152491 2082150
2069743	1894849209	PCI-MSI-X	eth1-rx-2		
178:	2445302424	1728604344	843543689	232630412	4365207 4219576
4031133	502923	PCI-MSI-X	eth1-rx-3		
186:	16761	325082928	712828704	249179575	2839276 2725962
2571980	255465	PCI-MSI-X	eth1-tx-0		