操作系统研讨课实验

Project 5: Device Driver

- 王华强
- 中国科学院大学
- 2016K8009929035
- wanghuaqiang16@mails.ucas.ac.cn

- 文档版本: 2018.12.19
- 程序版本: lab5_passed_vm_unstable

注: 提交版本中各进程为使用虚地址空间的用户进程. 若要在板上取得稳定的测试结果(虚存不稳定的原因参见下文), 请将所有测试进程切换为内核进程.

---

1. 目录

---

2. 网卡驱动设计

2.1. 接收描述符

接收描述符的配置如下所示.

```
uint32_t init_desc_receive(void *desc_addr, void *buffer, uint32_t bufsize, uint32_t pnum)
{
    int cnt = 0;
    // Fst Tx desc
    uint32_t start_addr = (uint32_t)desc_addr;
    uint32_t addr = (uint32_t)desc_addr;
    // uint32_t last_addr=desc_addr_now;
    os_assert(pnum > 0);
    // os_assert(pnum <= DESC_NUM);

    //Not the last one
    while (++cnt < (pnum))
    {
        ((desc_t *)addr)->tdes0 = 0x00000000;
        ((desc_t *)addr)->tdes1 = 0 | (1 << 24) | (1 << 31) | (bufsize & 0x7ff);
        ((desc_t *)addr)->tdes2 = ((uint32_t)buffer + (cnt - 1) * bufsize)& 0x1fffffff;
        ((desc_t *)addr)->tdes3 = (addr + DESC_SIZE)& 0x1fffffff;
        addr += DESC_SIZE;
    }

    //The last one
    ((desc_t *)addr)->tdes0 = 0x00000000;
    ((desc_t *)addr)->tdes1 = 0 | (1 << 25) | (0 << 31) | (bufsize & 0x7ff);
    ((desc_t *)addr)->tdes2 = ((uint32_t)buffer + (cnt - 1) * bufsize)& 0x1fffffff;
    ((desc_t *)addr)->tdes3 = (start_addr)& 0x1fffffff;
    addr += DESC_SIZE;

    return start_addr;
}
```

在正常的网络传输处理中, 初始化了64个接收描述符形成环形链表. 在高速传输模式下, 接受描述符的数目增大到256个.

接收描述符的各位域设置如下:

位置	内容
tdes0	0x0
tdes1	(1 << 24) 环形链表的非最后一个描述符 | (1 << 31) 不触发网卡中断 | (bufsize & 0x7ff) 缓冲区大小
tdes2	接收缓冲区物理地址
tdes3	下一个描述符的物理地址

最后一个描述符设置如下:

位置	内容
tdes0	0x0
tdes1	(1 << 25) 环形链表的最后一个描述符 | (0 << 31) 接收完成触发网卡中断 | (bufsize & 0x7ff) 缓冲区大小
tdes2	接收缓冲区物理地址

位置	内容
tdes3	下一个描述符的物理地址

2.2. 发送描述符

发送描述符的配置如下所示.

```
uint32_t init_desc_send(void *desc_addr, void *buffer, uint32_t bufsize, uint32_t pnum)
{
    sys_move_cursor(1,5);

    int cnt = 0;
    // Fst Tx desc
    uint32_t start_addr = (uint32_t)desc_addr;
    uint32_t addr = (uint32_t)desc_addr;
    os_assert(pnum > 0);
    // os_assert(pnum <= DESC_NUM);

    //Not the last one
    while (++cnt < (pnum))
    {
        ((desc_t *)addr)->tdes0 = 0x00000000;
        ((desc_t *)addr)->tdes1 = 0 | (0 << 31) | (1<<30) | (1<<29) | (1 << 24) | (bufsize & 0x7ff);
        ((desc_t *)addr)->tdes2 = ((uint32_t)buffer)& 0x1fffffff;
        ((desc_t *)addr)->tdes3 = (addr + DESC_SIZE)& 0x1fffffff;
        addr += DESC_SIZE;
    }

    //The last one
    ((desc_t *)addr)->tdes0 = 0x00000000;
    ((desc_t *)addr)->tdes1 = 0 | (0 << 31) | (1<<30) | (1<<29) | (1 << 25) | (1 << 24) | (bufsize & 0x7ff);
    ((desc_t *)addr)->tdes2 = ((uint32_t)buffer)& 0x1fffffff;
    ((desc_t *)addr)->tdes3 = (start_addr)& 0x1fffffff;
    addr += DESC_SIZE;
    return start_addr;
}
```

同上, 初始化了64个发送描述符形成环形链表.

接收描述符的各位域设置如下:

位置	内容
tdes0	0x0
tdes1	(1<<30) 当前 buffer 包含
的是一帧数据的最后一段 | (1<<29) 当前 buffer 包含	
的是一帧数据的第一段 | (1 << 24) 环形链表的非最后一个描述符 | (bufsize & 0x7ff) 缓冲区大小	
tdes2	发送缓冲区物理地址
tdes3	下一个描述符的物理地址

最后一个描述符设置如下:

位置	内容

位置	内容
tdes0	0x0
tdes1	(1<<30) 当前 buffer 包含
的是一帧数据的最后一段 | (1<<29) 当前 buffer 包含	
的是一帧数据的第一段 | (1 << 25) 环形链表的最后一个描述符 | (bufsize & 0x7ff) 缓冲区大小	
tdes2	发送缓冲区物理地址
tdes3	下一个描述符的物理地址

2.3. 环形链表和线性链表

此实验中使用环形链表. 一方面, 环形链表作为传输描述符在各种系统中应用的比较广泛. 另一方面, 使用环形链表可以使得DMA"当前描述符"寄存器(18/19)在完成一组描述符之后返回到起始地址, 方便重复利用发送描述符.

2.4. 检查数据到达网卡的手段

此实验中实现了多种同步方式:

2.4.1. 轮询

由CPU不断轮询传输描述符, 来判断传输是否完成. 这种做法导致某个进程不停进行轮询操作, 比较低效.

2.4.2. 有间隔时间的轮询

中间步骤(TEST2)"在时钟中断中检查收包情况"实际上是一个有间隔时间的轮询, 在间隔时间内进程放弃CPU控制权. 这种情况下降低了性能浪费, 但是在网卡收包完毕时不能马上做出反应.

2.4.3. 网卡中断

通过配置网卡中断, 可以实现在网络传输刚刚完成时就触发网卡中断. 在快速传输时, 通过网卡中断发生后立即唤醒等待网卡中断的进程, 可以实现及时的收包后处理(分配新的缓冲区, 填充新的描述符).

2.5. 网卡中断设计

在原有的时钟中断路径上添加了简单的判断. 因为此系统不准备支持大量的设备中断, 因此将网卡中断的相关处理直接固定在中断处理流程中. 通过读取CP0_CAUSE寄存器和INT1_SR寄存器来实现对网卡中断的判断.

```
    uint32_t ip=cause>>8;
    ip&=0x000000FF;
    if(ip&(1<<7))//ip[7]==1
    {
        irq_timer();
        return;
    }

    if(ip&(1<<3))//ip[3]==1
    {
        if((*(uint32_t*)INT1_SR)&(0x00000001<<3))
        {

            irq_mac();
        }
        // printk("invalid interrupt:%x\n",status);
        // panic("no_irq_mac");
        return;
    }
```

---

3. 高效收包实现

3.1. 可行的加速策略

实现快速收包的核心是: 在一组包接收完成之后尽快开始下一组包的接收. 减少中间重新配置描述符, 重新启动DMA传输所用的时间. 收包过程中浪费的时间主要在于:

1. 其他同时运行的进程占用的时间
2. 系统调用, 时钟中断需要额外执行大量指令.

为了实现节省时间的目的, 大致有以下思路:

- 提高收包进程的优先级.
- 减少陷入内核的次数.
- 减少重填描述符操作所花的时间.

更具体一点, 可以采用以下手段:

3.1.1. 增大时间片

通过增大时间片, 可以减少时钟中断所占时间的比重, 从而有更多时间执行收包相关指令. (减少陷入内核的次数)

同时也可以在时间片配置时给快速传输进程配置更大的时间片. (提高收包进程的优先级)

3.1.2. 禁用虚存(切换到内核进程)

通过在快速传输进程运行时禁用虚存, 可以避免TLB例外和SWAP守护进程占用太多时间. (减少陷入内核的次数)

3.1.3. 立即唤醒指定进程

声明 `do_immrun(queue_t*, pcb_t*)` 函数. 这个函数的作用是"立即唤醒并切换到指定进程". 这个功能也可以通过配置进程的运行优先级实现. (提高收包进程的优先级)

3.1.4. 增大描述符和缓冲区大小

通过一次处理更多描述符来降低网卡中断的频率. (减少陷入内核的次数)

3.1.5. 优化网卡中断处理流程

精简相关代码, 尽量避免io函数(如printf).

3.2. 具体实现

本实现中实际使用的策略如下:

3.2.1. 增大描述符和缓冲区大小

增大一次处理的描述符到256个. 这个方法可以使得性能从0.7Mb/s提升到约0.9Mb/s

3.2.2. 禁用虚存(切换到内核进程)

在进行快速传输时不开启虚存机制. 这个方法可以使得性能从0.9Mb/s提升到1+Mb/s

---

4. 调试相关问题

4.1. 网卡中断的初始化问题

为避免无限进入网卡中断, 网卡初始化时需要进行以下配置:

```
    *(uint32_t *)INT1_CLR = 0xFFFFFFFF;
    *(uint32_t *)INT1_POL = 0xFFFFFFFF;
    *(uint32_t *)INT1_EDGE = 0;
```

~~感谢刘蕴哲, 蔡昕同学踩坑~~

4.2. 寄存器配置的顺序问题

部分DMA寄存器配置需要按照顺序进行, 否则会导致模拟器通过上板不通过

~~感谢资威同学踩坑~~

4.3. 描述符总线对齐

实验中使用的开发版总线位宽为64bit, 而传输描述符是默认4字节对齐的. 因此务必确保传输描述符在内存中的地址为8字节对齐, 以使得DMA可以正常读取.

4.4. 虚存释放的问题

这部分属于之前的设计中遗留的问题. 暂时只在上板时复现, 且暂时没有定位到问题. 怀疑与虚存及网卡中断有关.

4.5. 虚存算法的补充

在之前进行虚存开发的过程中, 由于不知道 `tlbr` 指令可以读取TLB中的内容, 导致虚存算法的设计受到了很多的限制... 使用 `tlbr` 指令可以实现更合理的虚存管理算法(比如跨进程的调页).

---

5. 参考文献

- [0] [LAB5实验说明]
- [1] [龙芯2F手册]
- [2] [龙芯1C手册]
- [3] [刘蕴哲,蔡昕踩过的坑]
- [4] [资威踩过的坑]
