《深入理解Linux网络》

- 第2章 内核是如何接收网络包的
  - Linux的准备
    - 创建ksoftirqd内核线程
    - 网络子系统初始化，为NR_SOFTIRQ(softirq_action)数组添加
    - 向中断控制器，注册NAPI 原用的 poll 函数
    - 启动网卡，并为网卡创建RingBuffer
  - 动态数据的到来
    - 网卡将数据DMA到RingBuffer中
    - 发起硬中断通知，CPU进入硬中断处理，长度优先级软中断
    - ksoftirqd内核线程响应入软中断处理函数
    - 网卡收到的数据报到达socket对应的队列中
    - 通过同步阻塞或者epoll等方式告知用户进程进行写作
- 第3章 内核是如何与用户进程协作的
  - 同步阻塞
    - 用户进程
      - 创建socket对象（通过socket系统调用，或者accept接收）
      - 调用recvfrom系统调用接收数据
      - 到socket接收队列中查看是否有数据，有则接收并返回
      - 如果数据未到达，则休眠进程阻塞，让出CPU
    - 硬件与内核处理
      - 网卡将数据DMA到RingBuffer中
      - 发起硬中断通知，CPU进入硬中断处理，长度优先级软中断
      - ksoftirqd内核线程响应入软中断处理函数
      - 网卡收到的数据报到达socket对应的队列中
      - 唤醒阻塞socket等待队列上阻塞的进程，并将原网络数据入内核队列
  - 多路复用之epoll
    - 用户进程
      - 调用epoll_create创建epoll对象
      - 通过epoll_ctl为epoll添加需要监听的socket，可以添加很多个，注册ep_poll_callback回调
      - 调用epoll_wait等待可阻塞的socket上的可读、可写等事件
      - 如果有事件需要处理，则一直处理
      - 直到没有事件需要处理，或者时间片到达，释放出CPU
    - 硬件与内核处理
      - 网卡将数据DMA到RingBuffer中
      - 发起硬中断通知，CPU进入硬中断处理，长度优先级软中断
      - ksoftirqd内核线程响应入软中断处理函数
      - 网卡收到的数据报到达socket对应的队列中
      - 调用socket等待队列中阻塞的进程的ep_poll_callback函数
      - 将socket相关的epoll-item加到就绪队列中（等待用户进程来取）
      - 看一下epoll对象上有没有进程在阻塞，没有则不唤醒，有的话唤醒一下
- 第4章 内核是如何发送网络包的
  - Linux协议栈
    - 创建RingBuffer环形队列
  - 用户进程发送操作：长期消耗CPU的时间
    - 调用send系统调用
    - 将要发送的数据拷贝到内核态
    - 判断是否需要立即发送，若不需要立即发送数据就放到了队
    - 如果需要立即发送则在协议栈中进行TCP、IP等包头的封装
    - 在IP层进行路由查询，判断要使用loopback设备发送
    - 在TCP层进行拥塞控制，判断是否可以发送
    - 根据子系统协议调用下一层的xmit函数并继续发送
    - 通过网络设备子系统将包添加到发送队列高级发送
    - 自己发送，或者可能被加到中断来发送
  - 软中断发送：当用户进程系统态CPU用完时间片，软中断发送，会消耗CPU的时间
    - 获取发送队列
    - 调用网卡驱动，将skb_xmit_frame进行发送
  - 发送RingBuffer内存回收
    - 网卡发送完成后触发硬中断
    - 硬中断触发NET_RX_SOFTIRQ类型软中断
    - 进入网卡驱动的回调函数igb_poll，igb_clean_tx_irq
    - 调用dev_kfree_skb释放
- 第5章 深度理解本机网络IO
  - 本机发送过程
    - 调用send系统调用
    - 将要发送的数据拷贝到内核态
    - 判断是否需要立即发送，若不需要立即发送数据就放了吧
    - 如果需要立即发送则协议栈中进行TCP、IP等包头的封装
    - 在IP层进行路由查询，判断要使用loopback设备发送
    - 将要发送的数据送入队（等待软中断来触发）
    - 软中断执行，调用网卡设备的xmit函数process_backlog
    - 取queue队列上面下来的数据包
    - 将数据进行协议栈的处理
    - 处理完的skb socket接收队列
  - 本机接收过程
    - 调用recvfrom系统调用
    - 将要接收的数据拷贝到内核态
    - 判断是否需要立即发送，若不需要立即发送数据就放了吧
    - 如果需要立即发送则协议栈中进行TCP、IP等包头的封装
    - 在IP层进行路由查询，判断要使用loopback设备发送
    - 将要发送的数据送入队（等待软中断来触发）
    - 软中断执行，调用网卡设备的xmit函数process_backlog
    - 取queue队列上面下来的数据包
    - 将数据进行协议栈的处理
    - 处理完的skb socket接收队列
- 第6章 深度理解TCP连接建立过程
  - 深入理解listen
    - listen主要工作就是创建半连接、全连接队列
    - 全连接队列长度 = min(backlog, net.core.somaxconn)
    - 半连接队列是由是打开syn_cookie来解决的
  - 深入理解connect
    - 选择一个可用端口
    - 如果端口未被使用，则选择该端口
    - 如果端口被用，但存在的连接所建立的连接四元组并不完全一致，则该端口仍然可用
  - 完整TCP连接建立过程
    - 客户端发起第一次握手
      - 客户端调用connect进入内核态
      - 选择一个可用的端口
      - 初始新的套接字结构（242字节+至少，1120字节）
      - 发出syn握手包
    - 服务端响应第一次握手
      - 服务端收到第一次握手包
      - 判断半连接队列是否满了，满了且没开syn_cookie则丢弃
      - 将该握手请求信息记到半连接队列中（包含syncookie计算）
      - 判断全连接队列是否满，满的话丢弃
      - 初始新的套接字结构
      - 发出synack第二次握手包
    - 客户端响应第二次握手
      - 客户端收到服务端的synack
      - 调用connect状态的套接字结构
      - 修改自己的socket状态为ESTABLISH，打开相应的对象
      - 发出第三次握手包
    - 服务端响应第三次握手
      - 服务端收到第三次握手包
      - 创建一个新的子sock对象出来
      - 删除相应的半连接队列元素
      - 添加到sock对象到连接队列
      - 等待客户进程来accept把连接取走
  - 常见异常TCP连接问题
    - connect连接超时阻塞很久时间，导致CPU消耗上涨
      - 保持长连可用的端口范围
      - 使用非连接来减少握手的握手
    - 握手丢包引起的握手失败，伴随而来的重传导致的延迟
      - 加大全半连接队列
      - 打开syn_cookie
      - 应用程序调用accept
      - 调整重试参数如net.ipv4.tcp_syn_retries和net.ipv4.tcp_synack_retries
      - 使用长连接来减少握手的握手
- 第7章 一条TCP连接消耗多大的内存
  - Linux 内核如何管理内存
    - 第一步：将内存条按CPU划分为多个node
    - 第二步：把每个node划分成不同的zone，每个zone下都有许多page（4KB大小的页面）
    - 第三步：每个zone下的page通过伙伴系统来进行管理
    - 第四步：slab机制来管理所有内核对象（slabinfo伙伴系统申请可用pages）
  - TCP 连接相关内核对象开销：总共大约3.3KB左右
- 第8章 一台机器最多能支持多少条TCP连接
  - Linux最大文件描述符限制
    - 系统级别：fs.file-max
    - 进程级别：fs.nr_open
    - 用户级限制：soft nofile
  - 客户端端口65535耗尽问题的方法
    - 一台Linux上可以配置多个ip
    - 如果不配置多ip，那么一个端口也可以用于连接不同的server
- 第9章 网络性能优化建议
  - 网络请求优化
    - 尽量减少不必要的网络IO
    - 尽量合并网络请求
    - 能否考虑调用机器可以能异步的一些
    - 内网调用不要使用外网域名
  - 接收过程优化
    - 调整网卡RingBuffer大小
    - 多队列网卡RSS调优
    - 硬中断合并
    - 软中断budget调整
    - 接收合并处理
  - 发送过程优化
    - 控制缓冲区大小
    - 减少内存拷贝次数
    - 零拷贝
    - 多队列网卡RPS调优
    - 本机间尽可能不使用IP协议并以域名或者地址开销
  - 内核与应用协作方式优化
    - 少用同步阻塞的编程模型
    - 使用协方编程的异步模型
    - 使用kernel-bypass旁路技术
  - 操作系统优化
    - 配置内核的端口范围
    - 客户端绑定最好指定bind
    - 小心非连续、全连接队列溢出
    - 适当调小握手重试
    - 打开FIO
    - 保持内核可以稳定开上限
    - 如果握手频繁，最好用长连接代替
    - TIME_WAIT优化
      - tcp_max_tw_buckets
      - tw_reuse/tw_recycle(内核4.12中已取消)
- 第10章 容器网络虚拟化
  - veth设备对：类似loopback回环设备，区别是veth是一对儿虚拟的设备，一个发，另一个设备上就能收到数据
    - 类似loopback回环设备
    - 一个veth对应两个虚拟设备
    - 一个发，另一个设备上就能收到数据
  - 网络命名空间
    - 一台Linux上可以创建多个网络命名空间
    - 每个网络命名空间都是一个逻辑上独立的协议栈
    - 每个网络命名空间有属于自己的路由表、socket、网卡等网络资源
    - 每个物理网卡、虚拟设备都可以设置归属到某个网络命名空间下
  - 虚拟交换机bridge
    - veth是一头可以“插”物理和bridge上
    - bridge可以将不同网络命名空间中的veth设备打通
    - 让两台veth可以相互通信