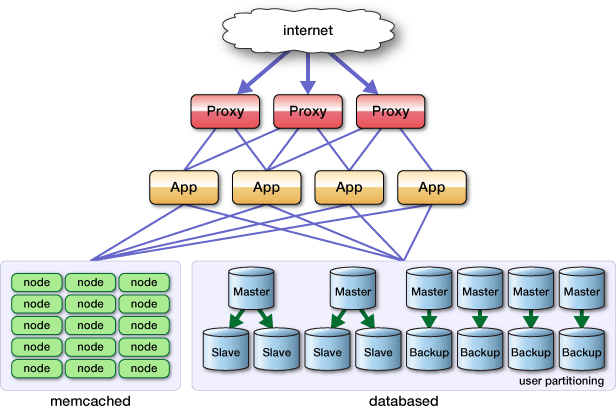
Memcached & kmemcache

1. 动机

对于C10k类问题，如何尽快的响应sock I/O？

Memcached服务器部署在廉价PC机，且只做cache一件事情，如何发挥其最大的性能？

1. Kmemcache是什么

Memcached 是一个跨平台的高性能分布式内存对象缓存系统，用于动态Web应用以减轻数据库负载。Kmemcache是基于memcached\_v1.4.15开发的，可以说是memcached的linux内核版本。  


图表 1 (memcached)

1. Memcached的实现机制
   1. libevent

libevent是一个事件触发的网络库，支持网络IO、定时器、信号三种事件类型，适用于windows、linux、bsd等多种平台，内部使用select、epoll、kqueue等系统调用管理事件机制。libevent事件处理流程如下：

图表 2 （libevent）

* 1. 线程模型

memcached的网络I/O事件处理主要一个主线程和多个工作者线程管理，每个线程对应libevent的一个事件池，主线程监听客户端的建立连接请求后创建新的连接，再派发到一个工作者线程处理读写事件，如下图：

图表 3 （memcached threads）

1. Kmemcache的实现机制
   1. 内核上下文

内核代码可工作于多种上下文下： NMI context、interrupt context、softirq context、atomic context、process context。内核中处理用户态进程的代码都工作在process context下，如epoll实现。

* 1. 网络堆栈概要

对于网络的正式介绍一般都参考了 OSI（Open Systems Interconnection）模型，但Linux 中基本网络栈实现基于四层模型，其对应关系如下图：



图表 4 (网络堆栈)

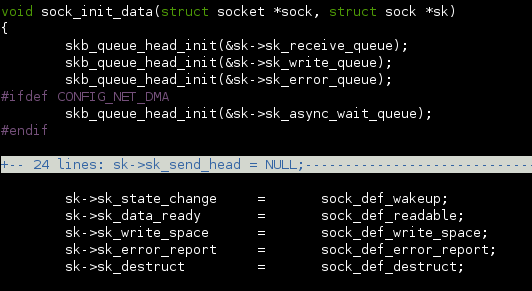
* 1. 内核套接口层

套接口最初由BSD 4.2（1983）引入，现已成为一个通用的网络应用程序编程接口，收到所有操作系统的支持。套接口层位于应用程序和协议栈之间，对应用程序屏蔽了与协议栈相关实现的具体细节，将应用程序发给的与协议无关的请求映射到与协议相关的实现，由此为应用程序提供了一个专门访问网络和进程通信的通用接口。内核套接口位于系统调用与网络栈之间，是内核提供网络服务的最上层接口，如下图：

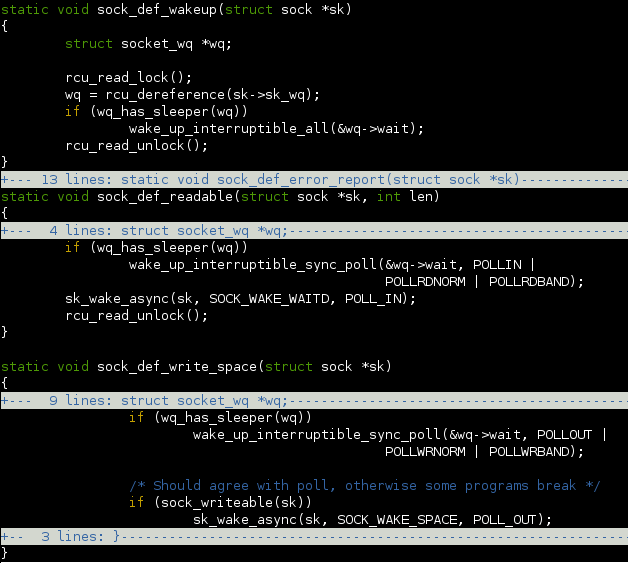


图表 5 （系统调用、套接口层、网络堆栈之间关系）

内核在套接字相关的对象时，会初始化套接字的几个回调函数（如图6），这些回调函数被网络堆栈代码调用，用于通知网络I/O事件（如图7）。



图表 6 （设置内核套接字回调函数）



图表 7 （内核套接口默认回调函数）

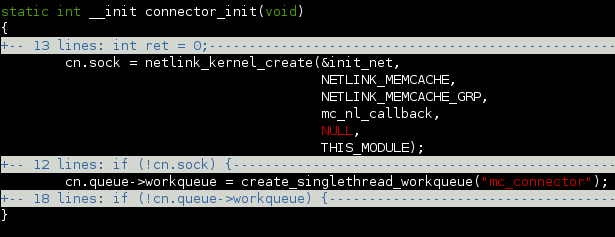
* 1. kmemcache总体设计

kmemcache服务器共有三个子模块：connector通过netlink机制，提供通用的用户空间与内核空间的数据交换服务；umemcached是个交互程序，提供kmemcache需要的初始化参数及环境变量等信息，运行时通过netlink socket连接到connector服务器，同步等待内核事件；kmemcache初始化时通过connector向umemcached发送获取初始化配置数据请求，之后独立运行，与客户端通过memcached通信协议，提供数据缓存服务。



图表 8 （kmemcache server）

* + 1. connector



图表 9 （connector init）

connector初始化时注册netlink服务，设置回调函数为mc\_nl\_callback，并创建一个工作队列（如图9）。对外提供的服务有：

* mc\_get\_unique\_val
* mc\_put\_unique\_val
* mc\_add\_callback
* mc\_del\_callback
* mc\_send\_msg\_sync
* mc\_seng\_msg\_timeout

connector提供了user/kernel通信的netlink机制简单封装，以kmemcache初始化为例分析其使用，见图10。



图表 10 （kmemcache 初始化）

* + 1. umemcached
    2. kmemcache
* 内存分配机制

kmemcache的内存分配器被设计成尽量减少系统的页间和页内碎片，此外在x86\_64架构下，直接通过伙伴系统获得的页面会被缓存，直到一个阈值或内存不足时释放。分配策略见图11，对外提供的服务如：

* alloc\_buffer
* realloc\_buffer
* free\_buffer



图表 11 （alloc policy）

* 工作队列

kmemcache初始化时，根据umemcached的设置，创建一个派发队列，多个工作者队列（如图12）。



图表 12 （work queues）

* 服务套接字

kmemcache初始化时，设置tcp listen socket和unix socket的回调函数，内核网络堆栈通过这些回调函数，将网络I/O事件放入上文描述的派发队列；对于udp socket服务，则直接放入工作者队列（如图13）。



图表 13 （server socket）

* 1. kmemcache请求处理过程（tcp为例）

前面说过，创建tcp 监听套接字时，设置了其回调函数，传输层收到网络层的请求后，会调用这些函数，将网络I/O事件派发到dispatcher queue，派发队列内核线程将创建新的连接请求，以轮询算法推送到一个worker queue后，结束本次处理；此后，工作者被唤醒，对应的内核线程依次处理队列中的请求（如图14）。



图表 14 （请求处理过程）

* 1. kmemcache优缺点
     1. 优点
* user/kernel切换
* sock I/O event based
* zero copy
* TLB缓存命中率
  + 1. 缺点
* Kernel panic
* 内核有限线性地址空间
* 单一进程空间
* 影响整个系统性能

1. 实验对比
   1. 实验环境

我们将cache服务器同时部署在一台物理机上，客户端在另一台物理机上。

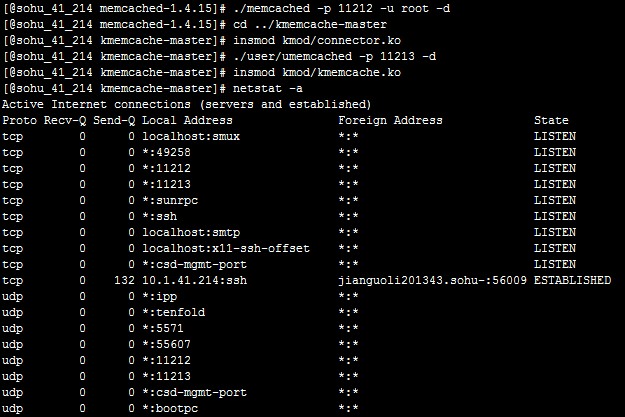
服务器：Xeon E5410 2.33GHz RAM(8G), redhat 5.3, linux-2.6.35.14

客户端：DELL E6230 laptop中虚拟机

测试工具：libmemcached中的memslap（set操作），kmemcached自带的memcached-tool （dump操作）

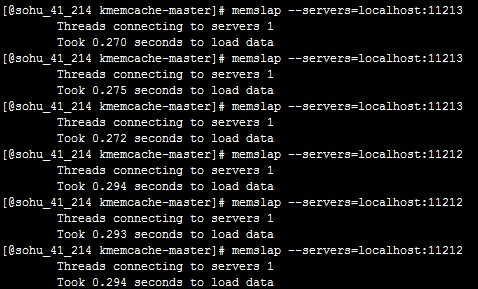
* 1. 测试过程

启动memcached服务器，端口号为11212，kmemcache的服务端口号为11213，如下图所示：



图表 15 （memcached & kmemcache）

使用memslap命令，每次进行10000次”set”操作：



图表 16 （在服务器端执行set）

使用”time –p memcached-tool ip:port dump”执行”dump”操作:

C:\Users\jianguoli201343\AppData\Roaming\Tencent\Users\593014114\QQ\WinTemp\RichOle\NJMBFJ4A(@N51C${W{N$H$1.jpg

图表 17 （计算dump时间）

* 1. 测试结果

5.3.1 利用memslap分别向两个缓存服务器执行10次操作（每次执行1万次set操作），结果表明memcached具有较好的稳定性，如下表：

图表 18 （set）

5.3.2 利用memcached-tool对缓存服务器执行dump操作（get所有数据），结果表明memcached具有更高的稳定性和效率，如下表：

图表 19 （dump）

1. Kmemcache的将来

6.1 完善现有功能。

6.2文件系统及ssd缓存支持：现阶段的kmemcache通过buffer管理器分配内存，再构建slab分配器，将对象缓存在ram中，并用LRU算法删除超时的数据，当系统内存不足或达到一定阈值时，会将部分的page导出到swap分区，释放更多的线性地址和物理页面，对于x86\_32架构，kmemcache可使用的存储空间 < 896MB；如果可以提供一个ssd的mapper device缓存对象，并在上面构建一种文件系统，其中每个文件对应kmemcache的每种slab对象（如图20）， 这样每个slab可以使用所有的系统总线地址，如x86\_32架构下，拥有20种slab的kmemcache可使用的存储空间 >= 20 \* 4GB。



图表 20 （kmemcache roadmap）

6.3 永久存储？

1. Q & A

参考文献

1. <http://en.wikipedia.org/wiki/C10k_problem>
2. <http://memcached.org/>
3. <http://libevent.org/>
4. <https://github.com/jgli/kmemcache>