问题：实现单机处理 1000 万并发请求

解决思路：

在 C10K 以前 Linux 中网络处理都用同步阻塞的方式，也就是每个请求都分配一个进程或者线程。但增加到 10000 个请求时，10000 个进程或线程的调度、上下文切换乃至它们占用的内存都会成为瓶颈，需要从硬件和软件分别进行优化：

硬件上：

1、内存

假设每个请求需要 16KB 内存的话，那么总共就需要大约 15GB 内存[1]。

2、带宽

而从带宽上来说，20%活跃连接，每个连接 1KB/s 的吞吐量，总共也需要 1.6Gb/s 的吞吐量。还需要配置万兆网卡，或者基于多网卡 Bonding 承载更大的吞吐量。

软件上：

1、在单机上使用 epoll 去处理 I/O，构建一个非阻塞的 I/O 模型，是当前解决 C10K 问题的主流办法。
2、在此基础上，部署 lvs+nginx 做负载均衡，实现高可用的 webserver，或许可以解决 C10M 问题。

实践

1、修改 OS 默认全局、进程级的限制

绝大部分 Linux 操作系统，默认情况下不支持 C1000K，因为操作系统包含最大打开文件数(Max Open Files)的限制，分为系统全局的，和进程级的限制。如下图所示，当前全局最大打开文件数(Max Open Files)有 100 万,已经很多了，但是这台服务器还是无法支持 C1000K.

```
[root@VM-238-73-centos ~]# cat /proc/sys/fs/file-nr
928     0       1604644
```

[1] 数据来自：https://www.ideawu.net/blog/archives/740.html#q4

在/etc/sysctl.conf 文件中修改默认值:

```
#
# For more information, see sysctl.conf(5) and sysctl.d(5).
kernel.panic = 5
kernel.core_uses_pid = 1
kernel.msgmnb = 65536
kernel.msgmax = 65536
kernel.softlockup_panic = 1
kernel.printk = 7 4 1 7
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
kernel.sysrq = 1
net.ipv4.ip_forward = 1
fs.file-max = 1020000
net.ipv4.ip_conntrack_max = 1020000
net.ipv4.netfilter.ip_conntrack_max = 1020000
```

重启系统服务就可以生效。

进程限制上，如下图所示，当前 Linux 系统的每一个进程只能最多打开 6 万个文件.。

```
[root@VM-238-73-centos ~]# ulimit -n
65535
```

为了支持 C1000K，同样需要临时修改这个限制:

```
[root@VM-238-73-centos ~]# ulimit -n 1020000
```

2、Epoll 构建非阻塞的 I/O 模型

epoll 的相关系统调用:

int epoll_create(int size);

int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event);

int epoll_wait(int epfd, struct epoll_event *events, int maxevents, int timeout);

epoll 代码实现：详见附件.cpp 文件

epoll 的调用结果:

```
[root@VM-238-73-centos ~/c10k_test/epoll]# rm -rf  epoll.c
[root@VM-238-73-centos ~/c10k_test/epoll]# vi epoll.c
[root@VM-238-73-centos ~/c10k_test/epoll]# cc epoll.c
[root@VM-238-73-centos ~/c10k_test/epoll]# ls
a.out  epoll.c
[root@VM-238-73-centos ~/c10k_test/epoll]# ./a.out
open pipe: fd = -1
open pipe: fd = -1
epoll_ctl: Bad file descriptor
```

问题：epoll_ctl(kdpfd, EPOLL_CTL_DEL, events[n].data.fd,&ev)返回值一直是-1，应该是 epoll 不支持文件 fd。

3、nginx 实现负载均衡

由于软硬件限制，这里就只搭建两台 nginx 用于调度，目的在于感受 nginx 在实现负载均衡的强大功能。首先在系统上下载安装好 nginx。

1、修改 nginx 配置文件，新增 upstream 模块，采用加权轮询，即跟据配置的权重的大小而分发给不同服务器不同数量的请求。

```
    upstream myserver {
        server 9.134.238.73:8001 weight=2;
        server 9.134.238.73:8002 weight=1;
}
```

2、最后访问 9.135.154.236（本机 IP），我们发现，按照轮询的权重访问两次 8001 端口，再访问一次 8002 端口，即实现了负载均衡。