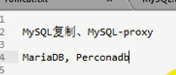
IO复用

昨天讲了MySQL复制 主从分离

中小企业 应用平台构建 后端使用多的是MySQL

尽管MySQL应用企业或者领域很多 但是逐渐走向封闭

建议多多关注



Perconadb是在5.6之后 转向MariaDBm

两年前 5.6的开发者之后20多个人 现在全球有200多个

MariaDB有一个硬伤 就是没有使用Innodb 这个存储引擎是Innodb base公司的 这个被Oracle收购的

MariaDB包含很多支持事务的存储引擎 但是 就是不包括Innodb

之前讲了MySQL的复制 HA集群 之后把所有的组合起来 组成大规模的Web的站点的应用

现在讲讲Nginx LNMP架构 Memcache的相关内容

如何使用memcache为Mysql提供缓存 如何使用memcache为php session提供缓存



Nginx的优势或者好处

简单看一下

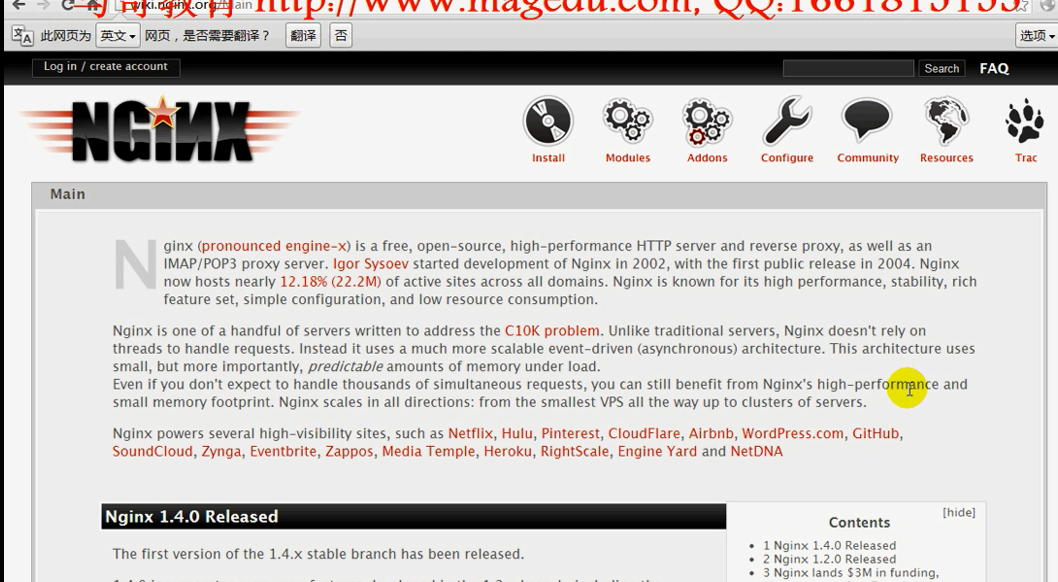
nginx.org

目前比较新的版本 是1.2.9和1.4.1

最新的分支就是1.4系列 1.2处于维护中



还有一个wiki站点





Nginx

 ---HTTP Server



Nginx是一个轻量级高性能的Web服务器 + 一个反向代理服务器

**任何一个代理服务器必须能够精确理解某种协议**

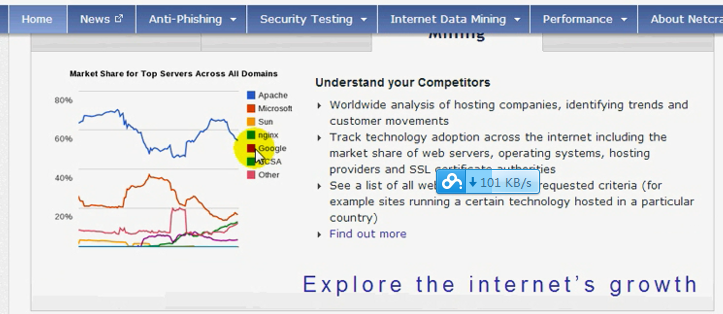
Mysql proxy就需要精确理解MySQL的各种查询语句 实现读写分离

对于反向代理 如果要代理某种 就必须要精确理解某种协议或者操作 事实上 nginx可以代理http和mail（smtp pop ixxx）的两种协议

Nginx早期就是用来设计为反向代理



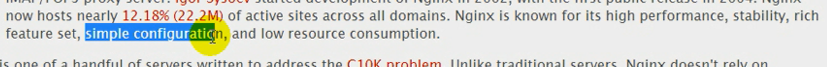




中小站点 通畅使用自己的IIS

Nginx从诞生到现在 一直是上升趋势

Nginx的特点：



高性能 稳定 功能丰富 配置简单 低功耗



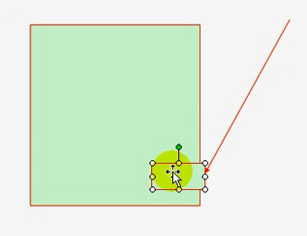
Nginx就是众多服务器中用来解决C10K问题的一个

C10 --- Connection

当我们的服务器并发达到10K会带来众多的问题

当一个服务器的主机有很多链接的时候 有一块网卡

每一个用户请求进来 工作在同一个套接字上面 如何在本地来响应外部主机的其他请求 什么模型？

 方块是网卡

**工作在一个主机上面的服务 无非就是 一个进程 或者一组进程 或者一组线程**

如何启用本地进程来响应用户请求？

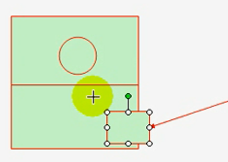
讲Apache的时候 如何实现 一个套接字上面的服务可以响应众多连接的请求

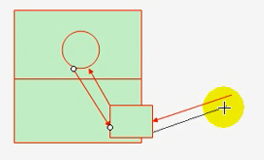
互联网刚诞生的时候 web用户非常少 开发一个服务器 完全可以工作在**阻塞同步模型**下

这样 一个互联网用户 启用当前进程来直接响应

监听在80端口的服务就是一个进程而已

请求由内核交由这样一个进程 进程处理完之后 通过网卡返回给用户

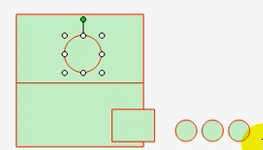
 圈代表进程



如果第二个用户来了 该怎么办？

阻塞模型下就意味着等待 这样 多个用户就产生了等待

这就是最简单的服务模型 非常简单 但是服务非常差



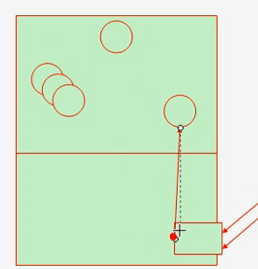
为了让多个用户同时得到相应 可以采用

Apache采用了pre-fork模型

Pre-fork如何解决？

在用户空间里面 有两种进程

在80套接字上面 有一个进程进行监听

 每通过一个网卡进来一个请求 就会被这个进程所响应到

之后 这个进程不直接进行处理这个响应 而是生成子进程 由这个子进程进行相应 然后返回给用户

做到了 “专进程-专用户”的这种方式

这种多用户相应得到了实现 这就是多用户进程模型

所以 一般早期 不会有很多进程 30-50

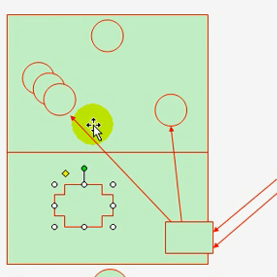
这个模型有一个巨大的问题：每一个用户请求 使用进程响应 进程自身开销很大 在OS运行的是一个单独在CPU上面执行的单位 必须分配内存资源 分配CPU时间片

这样进程的执行 势必带来很多的进程切换

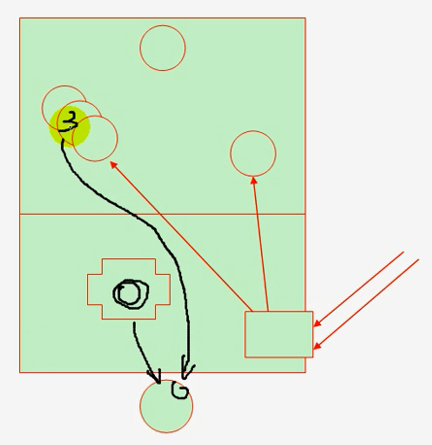
进程很多 每一个用户给一个进程响应 加入只有一颗CPU 这样一个时刻 只有一个请求被响应

但是 为了保证同时执行 那就是轮流切换 每个进程执行的时间仅仅执行有限的时间

切换如何完成？**内核**来负责处理进程切换

上面是用户空间 下面是内核空间 多边形是内核

这意味着 在某一时刻 内核内部有程序 运行自己的进程调度程序 观测每一个金发现下一次该轮到3号进程 唤醒3号进程 把3号进程载入到CPU的各种寄存器中 CPU的运行指针指向3号进程的地址空间 然后内核自己退出 转为睡眠状态 然后3号进程运行



3号进程运行结束了 给的5ms执行完成 然后内核又重新被唤醒 继续运行 将3号进程转入睡眠状态 然后就用自己的算法挑一个新的进程 这就是进程切换

如果进程切换非常频繁 结果会耗费大量时间 进程切换本身消耗的时间 对响应用户的请求没有帮助 所以 通产在内核空间消耗的时间通畅都是额外的内核开销 sys时间

所以多进程模型有这样一个缺陷

多进程模型下面 每一个进程都是一个独立运行的单位

多个用户请求很可能访问的是硬盘上面的同一个资源 比如打开同一个主页

如何打开的？

一个用户的请求接进来 某一个进程来响应 这个进程发现用户要求读一个页面 如何读？

系统调用 转入内核模式 这个进程处于阻塞状态 ---- 因为产生了IO调用

这个进程需要到硬盘上面读文件 但是进程本身又不能访问硬盘 所以 向内核发起请求 等待内核返回资源 但是 这个时候 这个进程没有响应完成这个请求 不能响应下一个 所以 这个进程进入睡眠状态 这样 就造成了阻塞 这叫**不可中断睡眠** 就是 你这个时候把这个进程唤醒 他也无法响应用户请求

Uninterruptable

所以就产生阻塞了 因为是因为系统调用 ---- IO产生的阻塞 所以等待IO完成 在CPU的时间上称之为waittime 转入阻塞之后 内核就负责到硬盘上取数据 数据取来之后 放到谁的内存里面？内核有内核内存 进程有进程内存

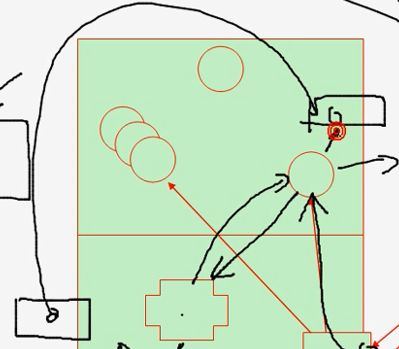
系统启动起来之后 这个内存是分成两段的 分别是内核空间和进程空间

内核加载到数据之后 先放到内核自己的内存中 也就是buffer或者cache中

之后 数据读取结束 要求进程访问这些数据 但是进程无法访问内核内存空间的数据 否则就乱套了

这时候 要把数据从内核内存空间复制到进程内存空间中

于是



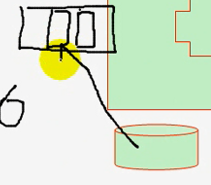
数据从硬盘加载到内存 我们的内存是分页的

这些文件要先被加载进来 内核内存是不分页的

每一个page 4K 一次IO读一个页面的数据 磁盘可以分成 1K 2K 或者4K的磁盘块

这样如果是1K的磁盘块 要读4个块 2K的内存块 要读两个块

这样 读进来 填满这样一个页面page



再读进来 填满第二个page 直到读取数据完成

有时候 不必读取所有的数据（数据可能太大）处理机制是别的方式 这里面不过多解释

这样 理解到 文件加载比较复杂

加入一个文件需要占据10个内存页 磁盘块每2K一个块 这样就是10\*4K/2k =20个磁盘块

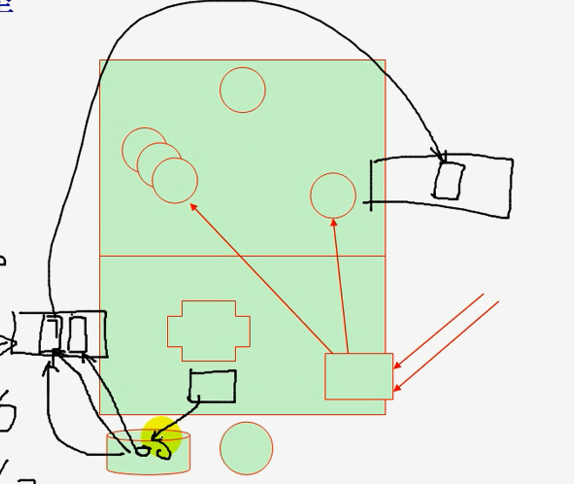
每一次读取哪个块 读完之后放到哪一个内存页中

这个需要CPU参与 CPU必须在内存中分配出一个内存空间

然后把数据加载进来

内核空间存储的数据是为了缓冲的 最终要加载到进程内存的内存空间中

读进来一段数据 就复制到进程的内存地址空间中



这个过程 如何选择空闲内存的页面 从哪里分配 要从磁盘哪些地方读取

内核操作的是文件系统 文件最终对应的数据一定是磁盘块 这个对应关系是驱动程序管理的 驱动程序运行 是内核的功能 意味着驱动程序的运行也是需要CPU

这些都需要内核的参与

加载一个文件的过程 我的内核空间很繁忙 为了尽可能降低内核对CPU的占用 让CPU运行更多的外部进程 现代很多硬件都是具有DMA的

DMA ---- Direct Memory Access直接内存访问

之后 CPU大概这样工作：

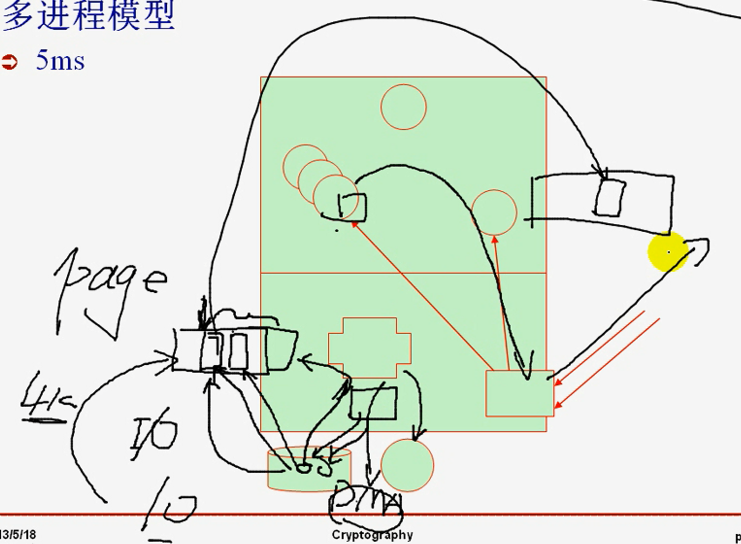
DMA模式下 需要10个页面 内核在自己的内存空间找一段连续的空间 连续的空间有一个起始地址 把这个起始地址给DMA 是主板上的一个芯片 是控制芯片 ---- 有访问内存的能力 --- 也就是有数据传输和占据系统总线的能力 ----- 有复制数据的能力 有控制数据总线 系统总线 和 控制总线的能力

由此CPU在从内存中分配找到这个起始地址+并决定读取哪些数据之后 就把总线的控制权交给DMA了 于是DMA指挥把磁盘中的数据加载到内存中 占据数据总线 还占据控制总线 比如发控制指令---读第几个磁盘块 然后占据数据总线 比如读到那个内存页中 所以 是DMA执行某些过程 这个时候 我们的CPU做什么？

比如有一些进程获取到了数据之后 跟网卡进行交互给用户的时候---- 需要占据数据总线么？

---- 不需要 只需要通过PCI总线和设备相连

所以 这时候 CPU就可以腾出时间 为已经准备好数据的进程服务 DMA指挥该做什么就做什么【这样不就是 我的CPU原本做的内核的工作 都交给了DMA处理 符合CPU为更多的外部进程工作】



如果DMA控制着 把所有的数据读入内存 DMA能控制这个进程来执行么？不能 当数据加载完成之后 DMA要产生一个中断 强行告诉CPU 你派给我的活完成了

此时 CPU立马产生中断 中断以后 刚才被CPU正在执行的进程就转入睡眠状态 此时内核加载好的数据 复制给外部请求的进程了 【也就是 DMA帮助完成了 硬盘数据到内存的一个工作 但是 数据从内核空间复制到外部线程空间的工作没能完成 还是后面的内核+CPU来完成】

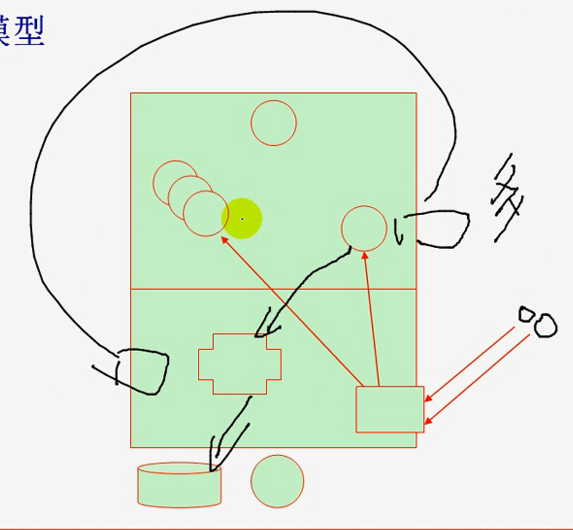
这样看来 这个机制非常复杂 所以开发OS内核非常复杂 涉及到许许多多复杂的仲裁机制

由此可以看出 一个进程

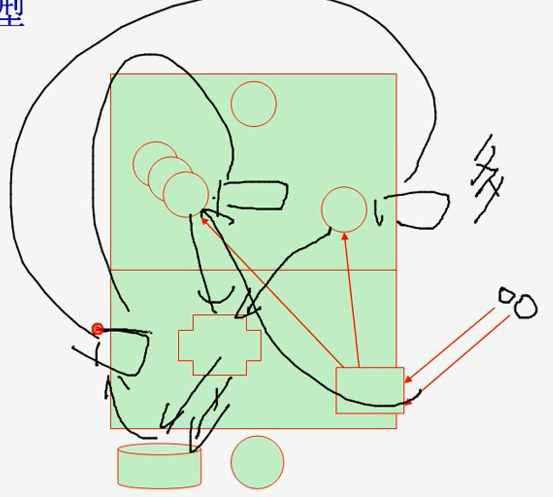
====== 回到我们的问题

多个进程需要访问同一个文件 结果是：

第一个进程IO ---- 加载文件->内核内存->进程内存



接下来第二个请求：



当然内核又加速机制 就是 我第一个读完的结果会缓存

第二个进程访问的时候 就是进程之间的复制过程

现在假设 有4个用户空间的进程访问我的主页 我的主页有2M的大小

请问：我的用户空间的内存空间 占据多少空间？要占据8M[因为不同的进程读取的进程内存空间的地址不一样] 实际上 我们2M能解决问题 但是不得不用8M 这就是多进程的缺点之一

------- 总结 ------

单进程模型：工作在阻塞模式下

多进程模型：每个进程响应一个请求

缺点 ---- 进程量巨大 🡪 进程切换次数非常多 -> 切换本身会消耗大量的资源

10K个进程进来 这样 从中选一个切换 就是很大的问题 无法处理10K问题 进程切换过多

---- 每个进程的地址空间独立 ->很多空间是重复的数据 -> 内存利用效率较低

这是多进程模型

如果不用进程 使用线程

在Linux上面 线程是进程内部的子单位 但是 遗憾的是 linux不支持原生态的线程 linux把所有的线程当成轻量级进程 windows和solaris支持原生态线程 但是linux不支持 管理上略有区别  叫LWP

在Linux上真正提供管理线程的线程库有很多 不同的线程库的管理对线程的管理所消耗的资源是不一样的

问题：

我们以线程的模式来讲解一下

所谓线程就是运行在进程地址空间的子单位

进程内部可以启动多个线程 就是多个执行流 线程也是并发执行流或者并行执行流

一个进程地址空间 分成了代码段区域 --- 一个程序要执行 要执行 有很多指令 要操作很多数据 程序就是指令+数据 所以下面有数据区

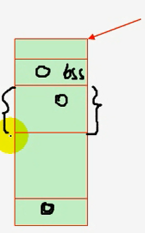
所以 有初始化为0的数据 以及 静态变量 在下下就是堆的空间 然后就是栈的空间（存放本地变量）

---- 在CPU中运行的就是指令

把指令的一条条执行 只不过用到数据的时候 就会去堆区 栈区 和 数据区获取数据

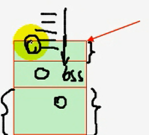
---- 这些堆区中的数据通畅都是打开的文件

---- 变量在最下面的示意图

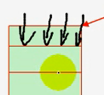


全局变量 BSS 在第二个格子

每一个进程的数据都是自我管理 所以在进程的时候 执行流的执行指令只有一个流水线



有了线程之后 可以并发执行 多个执行流

 这个执行流共享这些数据区域

这些线程 就是运行在进程地址空间的共享很多资源的运行的独立实体

所以 当某一请求来的时候 用一个线程来响应 这样 就是线程 进行系统调用 文件加载 而后放到数据区中

第二个请求来的时候 使用第二个线程来响应 如果访问的是同一个文件 这个文件已经存放在了这个进程的数据空间 这样 就是独占一份的数据 不会重复

这就是线程的好处

所以 基于线程来做 可以做到每个线程产生一个请求

好处总结一下：

\*\*线程要不要切换？ 线程也要切换 线程切换不可避免 因为线程执行也是独立执行实体 但是线程切换的代价小一些 因为数据是共享的 切换是轻量级的

进程管理的时候 到CPU执行的时候 需要进程切换 一个进程被切换出去 需要保存现场 切换进来 需要恢复现场（我有100个指令 执行到第70个 我的指令指针 还有指令处理要处理数据 这样我的数据还要再加到CPU的缓存中 一级缓存 二级缓存 一级指令缓存 二级指令缓存 还有就是寄存器 这些数据都必须要恢复回来 所以要大量操作）

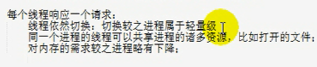
对于线程切换 由于很多数据都是共享的 所以 CPU对应的缓存的数据就可以不再需要切换了

如果一个进程间切换（线程切换）还好 到是 如果是线程间切换 需要切换众多资源

但不管怎么讲 同一个进程内部的线程切换 就是轻量级 ----- 可以共享进程的诸多资源

比如打开的文件 传入的信号等等

所以 多用户访问同一个文件的时候 在性能多线程的性能就会好很多 线程本身对内存的占用量变小 较之进程略有下降



但是无论如何 线程依然需要切换 所以 10K个请求 就会导致10K个线程

虽然线程可以并行执行 但是 和你当前的CPU的颗数很少 线程的优势就发挥不出来

如果是多核-🡪如果每一个线程在不同的CPU上 就有优势



这就是并行编程机制 可以更好的分配资源

除了web服务器进程意外 还有其他进程

比如来了10K个链接 难道都使用这个进程中的10K线程来做么？忙得过来么？

为了解决进程之间的资源争用

共享没错 但是 一个线程是写 另一个线程要读 能读么？必须等待写完

资源竞争依然存在

读是共享的 但是读写是不能共享的

比如一个资源正在被写 然后切换到另一个要读的线程来执行 但是 没有写完 这个线程不能读取 所以 又不能浪费CPU 所以这个时候 这个读的线程不得不被切换掉

读不了 要等一会 等有两种方式：忙等和闲等

忙等 --- 读不到 可以等一会 但是 CPU不允许空闲 因为CPU的时钟频率就在那不变 --- 只要CPU工作在一定的频率之下 无论用不用 都是一直走下去 ---- 无非就是白算 ---- 只要供电就好 ---- 所以 没有指令执行 这个线程就不能站在CPU上面 ---- 这是不允许的 ----- 所以 每隔1s来看看 ---

闲等 --- 直接再次切换出去 这就是闲等

一直处于过来看看 时间没消耗完 我就不走 这就是忙等

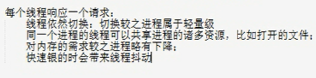
我的事情没干完 就切换退出 这就是自愿退出 这就是闲等

忙等通过自旋锁完成 ---- Spin Lock

何况如果线程过多 ---- 第一个是资源竞争激烈 另一个就是线程抖动

一个线程啥事没干就出去了 这就是线程抖动 这个是非常严重的

快速切换时 会带来线程抖动



所以10K个请求 显然不能用一个进程的多个线程来响应

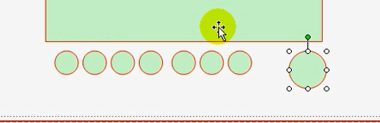
所以 还有一个模型就是 多进程-多线程模型

N个进程 每一个进程提供一部分线程

这样同一个进程内部 线程争用的情况不会太剧烈

多进程多线程就是 --- 每一个进程管理自己的进程 虽然有进程 线程切换 但是 有8个CPU

其中 留1颗出来



最右边的运行内核 左边的7个空闲出来 不运行其他

启动7个web进程 每一个进程绑定到一个CPU上面

这样 每一个CPU上面就不会有进程切换了 但是还有线程切换

原来的进程切换就没有了 这样做提高了系统的使用效率

操作系统要完成这种能力 要自己把CPU手动隔离出来 自己手动进行绑定

这样 还是存在线程切换 来10K的请求 还是比较吃力

下面就有了多线程多个请求 ---- 就是**一个线程响应多个请求** ----

如何做到？

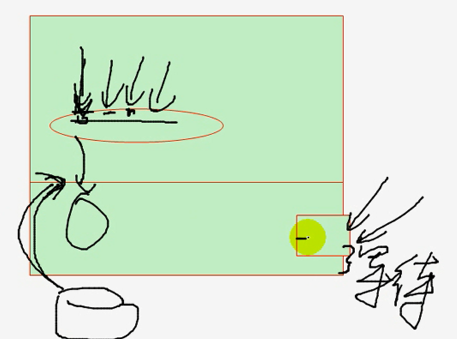
早先一个进程一个请求 或者 一个线程一个请求 之所以是一对一 就是IO不同 访问不同的磁盘IO

IO 就要去等待 这样客户端就要等待这个请求

问题是：一个线程响应多个请求的时候 过来多个请求 概念是每一个请求并发 都在一个线程上运行

这样这个请求的线程去系统调用 加载磁盘数据 完成之后 如何唤醒对应的线程呢（一个线程多个请求 怎么唤醒）？只要请求了资源 正在IO 这个线程就等待状态 ----- 踢出去 这时候处理别的线程 处理第三个时候 第一个的IO完成了

前面的线程IO完成之后 就要唤醒对应的线程 如何叫回来 但是网络连接却在内核里面



链接本身在网卡上进来的 和TCP-IP相关 TCP-IP相关的内容都是要由内核来处理的

很显然 线程要能连接到内核 把这个线程激活才行 如何激活？

实际上换一种方式解释：当来自一个线程多个IO

每一个IO交给内核来执行 内核准备好IO之后 --- 通知这个线程是哪个IO完成的？

IO两个：网络IO和磁盘IO 以磁盘IO理解

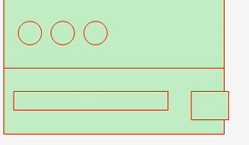
还有一个问题 激活这个线程之后如何响应给前端的链接呢？

所以 这个IO是两段的 磁盘IO和网络IO在磁盘量很大的情况下 如何解决？

磁盘IO也可能阻塞的

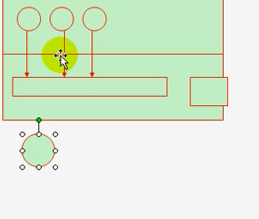
-----

可能有多进程 每个进程响应一个请求



磁盘IO的完成：每一个进程IO的时候 就会阻塞 就不能干别的事

同理：第二个也就这样 做不了别的事 就不能占用CPU 就会让出CPU的使用权



这时候 一旦内核准备好了一个IO 内核如何通知第二个进程 你这个IO准备好了？

或者说第二个进程如何知道这个IO准备好了？就算可以告诉第二个进程你的准备好了 但是 第二个进程如何感知我的IO好了呢？

所以内核需要输出一个数据结构：

放一排灯 谁的IO好了 谁的灯亮

来一个进程就放一个灯泡 亮了就准备好了

每隔一段时间 内核准备好一个 就会把所有的灯泡都处理一遍



所以这样一来 内核每一次要把整个数据结构向用户空间输出一遍 --- 不输出 这些进程无法理解 每一次都扫描一下IO文件的描述 然后把数据结构都输出到用户空间

这种模式叫select

Select最多支持1024个灯泡 再多 就运行不了了

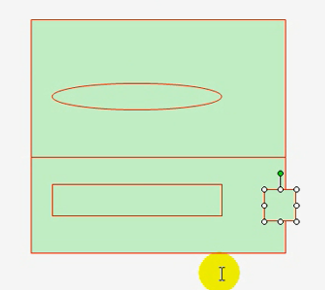
当然1024在早年量可以

所以这就是单进程 每一个进程响应一个用户请求面临的难题和在IO要解决问题 ---- 早起内核使用了算法 就是select算法 输出整个数据结构 来通知对应的线程

进程依然避免不了这个问题 --- 线程模型下 共享数据 这个进程描述符就变少 可能会响应多一点的用户请求 但是linux是轻量级的线程 所以 还是无法真的解决 只是可以共享数据 提高内存使用效率

线城带来的好处 就是响应速度提高了 数据共享了 内存使用效率提高了 但是 响应10K个链接还是无法满足的

所以就提高了一个线程响应多个请求



存在的问题：

多个用户请求

第一个用户请求的IO 进程要等待 要阻塞 这个线程就切换出去了 这样这个还是要这个线程来处理 第二个请求 ---- 但是由于刚刚被切换出去了 ---- 就不能处理第二个请求了 ----- 但是如果第二个没有请求IO 仅仅执行PHP程序 那么 一阻塞 就处理了

一个线程响应多个请求 一阻塞 那么原本要求这个线程处理的多个请求就都被阻塞出去了 ---- 怎么办？

很显然 请求一个IO的时候 不能阻塞

一个线程处理多个请求的时候 就算没完成 也不能阻塞 – 所以这个线程无非是 这个请求IO没完成 不能阻塞 那我就处理别的请求去了

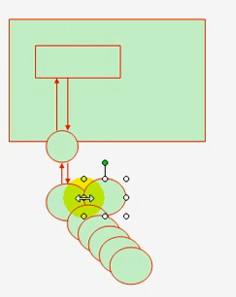
不能阻塞 一旦准备好了 说明你没在这等着 准备好了 怎么通知给你？

之前举过例子

就是饭店吃饭的例子

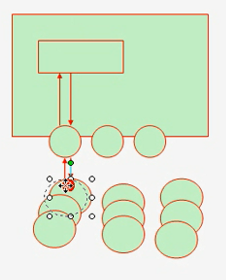
要了一碗饭 一个营业员接待了你 就有一个服务员 来了10个顾客

阻塞模型 第一个用户要打卤面 这个营业员要告诉后厨师父（相当于内核） 这个师父准备好之后 会告诉服务员 这个顾客的请求完成了 然后下一个



提高效率

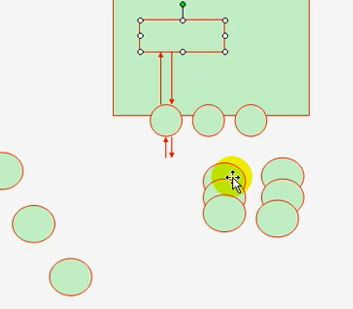
可以多派几个服务员 后厨很牛 可以多个锅



如果这时候 是阻塞模式 就是多队列的阻塞 一个队列完成 才能到下一个队列

非阻塞模型

第一个用户报号之后就可以去等待了 下一个去报号



某一碗饭做好之后 只有三个服务员 但是偶遇九个人 怎么办？

阻塞就是一个服务员只能接受一个客户 后厨做好

阻塞状态的时候 如果后厨准备好 我这个服务员阻塞了 就在这等着 所以 后厨准备好 我就可以感知到

非阻塞状态 ---- 一个线程多个请求 用户报号好 服务员给后厨报好饭之后 就直接处理下一个用户的请求 --- 这时候 后厨做好了 一放那 怎么办？

假设有一种模式通知了服务员 这就是非阻塞

这样 就是客户要一个东西 等待你给我 得到了 就退出 没得到 我就干点别的

第一个顾客报上饭之后 没有立即给后厨 而是多赞了几个顾客的 在给后厨

你点东西了 我就立刻给后厨 --- 这个就是同步通信 ---前端直接交给后端 否则 就是一部通信

异步和阻塞是两码事 和非阻塞也是两码事

-----

无论怎样 这几个用户都到一边等着 假设后面解决了非阻塞问题 要有办法告诉用户饭好了 怎么通知？

放一块电子屏幕 自己盯着电子屏幕等通知

为了让每一个用户知道自己的是否好了 每一个用户要来看自己的饭好没好 就要扫描一遍

这也是很痛苦的

我们可以简化一种机制 只通知好了 不通知没好的 这样显示的信息量就会小很多

通知好了的 要几遍？显示完之后 处理别的请求 但是如果这个顾客没有看见怎么办？

忙不过来 漏掉了 别人看不到 怎么办？

没看见 反正我尽到通知的义务了 不管你

如果 有的人没端走 我再通知一遍

这就是通知机制

这样 用户可以工作在异步模式下 该干啥就干啥 有服务员负责通知 可以都通知 也可以只通知好的 可以一次通知一遍

解决了多个用户同时过来请求 又都能让每一个用户都能知道自己的请求都完成响应的 就叫多路IO机制 --- 或者叫IO复用

一个线程响应多个请求 如何解决的？

1:28：21