云风的 BLOG

思绪来得快去得也快，偶尔会在这里停留

« 上次提到的阿瓦隆辅助工具 | 返回首页 | Xenonauts 中文化计划 »

对象到数字 ID 的映射

skynet 中使用了一个 hash 表结构来保存服务和 32bit 数字地址的映射关系。

一个 skynet 的服务，其实是一个 C 对象。在有沙盒的系统中，尤其是并行构架，我们很少直接用 C 对象指针来标识一个 C 对象，而是采用数字 id 。用数字做 handle 可以让系统更健壮，更容易校验一个对象是否还有效，还可以减少悬空指针，多次释放对象等潜在问题。比如，操作系统为了把用户程序隔离在用户态，像文件对象就是用数字 id 的形式交给用户程序去用的。

和操作系统通常管理文件句柄的方式不同，skynet 尽量不复用曾经用过的 id 。这样，当你持有了一个已经不存在的 id ，不太会误当作某个新的对象。（操作系统的文件 handle 很容易造成这样的问题：你记住了一个文件，在别的流程中关闭了它，然后又打开了新文件，结果复用了过去的 handle ，导致你操作了错误的文件。很多年前，我就在自己的线上产品中碰到过这样的 bug 。）

但是，一旦尽量不复用 id 。用简单的数组来做映射就很难了。必须使用 hash 表来保证映射效率。在 skynet 中我是这样做的：

每次分配新的 id 时，首先递增上次分配出去的 id （允许回绕，但会跳过 0 。0 被当成无效 id ）。然后检查 hash 表中对应的 slot 是否有冲突，若有冲突就继续递增。如果 hash 表的池不够用了，则成倍扩大池，并将内部的数据重新 hash 一次。

这样虽然会浪费一些 id ，但是可以保证 hash 表类的 key 永远不发生碰撞，这样可以让查询速度最快。hash 表的实现也相对简单一些。

我觉得这样的一个数据结构有一定的通用性，今天花了一点时间把 skynet 的这个部分单独抽出来，当成一个独立开源项目重新写了一遍。有兴趣的同学可以在 github 上查看。

这里的 struct handlemap 就是这样的一张 hash 表。而 handleid 目前被定义为 unsigned int ，可以在实际使用时定义为其它字长的整数。

一共提供了 5 个 API ：

```
struct handlemap * handlemap_init();
void handlemap_exit(struct handlemap *);
```

分别是创建和销毁 handlemap 结构，没什么好说的。和 skynet 的实现不同，这里做了一些简化。在销毁 handlemap 时，暂时没有手段去清理里面那些还没有删除的 id 。

我认为在实际使用时，可以通过外部手段去解决。

- handleid handlemap_new(struct handlemap *, void *ud); 就是为 ud 分配并绑定一个 id 。这个操作一般会成功，但在很少情况下会因为内存不足而失败。失败则返回 0 。 0 在整个系统中永远表示无效 id 。
- void * handlemap_grab(struct handlemap *, handleid id); 取出 id 对应的对象指针，并在内部增加对它的引用。如果 id 无效，那么会返回 NULL 指针。在实现时，采用了引用计数用来满足线程安全。
- void * handlemap_release(struct handlemap *, handleid id); 当使用完对象指针后，应该调用这个 api 归还 id 的引用。另外，如果希望删除这个 id 也可以调用它。无论是归还引用还是删除，都必须检查这个 api 的返回值。当返回值不为 NULL 时，表示返回的这个 ud 已经脱离了 handlemap 的管理，通常你需要删除这个对象。

这个数据结构的实现难点在于需要线程安全。在不同的线程中可以并发去 grab 对象，同时也可以安全的删除它们。

我使用了一个读写锁来保证线程安全：

在创建新 id ，或当一个 id release 的时候引用减到零，都会加上写锁，完成成 handlemap 的修改。

而获取（grab）或释放 （release ）id 时，则是加的读锁，这些操作是可以并发的。

btw, 和 skynet 不同，这份实现里我支持了 windows 的原子操作 api 。但没有仔细测试，如果有兴趣的同学可以试着用 VC 编译。

云风 提交于 April 10, 2015 02:03 PM | 固定链接

COMMENTS

我有过类似的实现，但是是通过数组+ID取模的方式实现O1的效率，因为只是指针数组，所以我的实现一开始就分配了固定的slot，每次新分配从当前最大的ID+1开始，查找空slot，找到后就返回ID，整个进程生命周期ID不重复。但是这份实现有一些限制，总ID数是编译时决定的，分配ID的途中会跳过非常多的ID，所以我的id是int64的。怕频繁分配和释放会达到最大的数值

```
#define channel_pool_put(p,i,c) (p)->channels[(i)%MAX_CHANNEL_POOL]=(c)
#define channel_pool_remove(p,i) (p)->channels[(i)%MAX_CHANNEL_POOL]=NULL
static channel *channel_pool_get(channel_pool *pool, channel_t i){
channel *chan = pool->channels[(i) % MAX_CHANNEL_POOL];
if (chan && chan->id == i){
return chan;
}
return NULL;
}
//查找空格，
static int find_empty_slot(channel_pool *pool){
int i=0;
for (i = pool->current_empty + 1; i != pool->current_empty; i++){
pool->maxid++;
if(channel_pool_get(pool,i)==NULL){
pool->current_empty = i;
return pool->maxid;
}
}
return -1;
}
```

Posted by: hongzi | (3) April 13, 2015 11:49 AM

推荐uuid~

Posted by: jichong | (2) April 13, 2015 11:11 AM

fork

Posted by: 天空 | (1) April 13, 2015 09:41 AM

POST A COMMENT

非这个主题相关的留言请到：留言本

名字：

Email 地址：

为了验证您是人类，请将六加一的结果（阿拉伯数字七）填写在下面：

URL:

记住我的信息？

留言：

（不欢迎在留言中粘贴程序代码）

提交