高并发认证服务器的一种实现方法①

樊扬轲

(沈阳师范大学 软件学院, 沈阳 110127)

摘 要: 本文介绍了一种高并发认证服务器基本功能, 具有并发、客户端与服务器双重认证的特点, 系统采用 epoll 和线程池技术来实现高并发功能,采用 RSA 和 DES 算法来实现信息传输的安全性, 使服务器具有并发处理能力和进一步提高了服务器网络安全认证性能. 通过完整设计实例测试结果表明,认证服务器在高负载的情况下运转正常. 该系统作为在短时间内同时处理来自客户端请求及认证的研究具有一定的现实意义.

关键词: 并发; 认证服务器; 线程池

An Implementation Method of High Concurrency Authentication Server

FAN Yang-Ke

(College of Software, Shenyang Normal University, Shenyang 110127, China)

Abstract: This paper introduces a kind of basic functions of authentication server, high concurrency authentication server has the characteristics of concurrent two-factor authentication of the client and server, epoll is adopted in the system and the thread pool technology is adopted to ensure that its high concurrency, RSA and DES algorithm is used to guarantee the security of information transmission, the server has concurrent processing ability and further improves the performance of network security authentication server. Through the complete design example test, results show that the authentication server runs normally in the case of high load operating. The system has certain practical significance to process synchronously from the client request and certification research in a short time.

Key words: concurrent; authentication server; thread pool

1 引言

目前, 数据的加密与认证广泛地运用在基于 http 协议的web服务器中比如网站的登录、网上支付等. 自从“棱镜”事件曝光后, 如何在越来越广的范围内保护我们的隐私, 成为了现在的热点问题. 一个高并发服务器是在短时间内同时处理大量的来自客户端请求的一个系统. 而现在常用的服务器如 WEB 服务器, 游戏服务器, 并发性都是有限的. 因此, 本文介绍的一种高并发功能的认证服务器, 具有高并发、客户端与服务器双重认证的特点, 使服务器具有高并发处理能力和进一步提高了服务器网络安全认证性能.

2 服务器框架

2.1 设计框架

传统的基于多进程和 select 机制的并发服务器由于受到系统资源和 Linux 内核版本的限制, 已经不能满足高并发的要求. 采用了 epoll 和线程池技术, 设计了一个高并发认证服务器. 服务器利用一个监听线程, 接受客户端的请求后放入连接缓冲区, 使用 epoll 技术来做事件触发, 再由线程池中的处理线程处理客户端发来的认证请求. 处理线程需要将客户信息解密后与存放在 MySQL 数据库中的信息进行比对, 从而完成认证过程. 其设计框架如图 1 所示[1,2].

2.2 工作流程

服务器的主要工作流程大致如下, 当启动客户端后, 首先应确保程序的单实例运行, 然后初始化运行日志, 当初始化完成后系统就会读取服务器的配置文件. 然后进行命令解析, 如果输入的参数为“-v” 将显示系统的版本号, 如果输入的参数为“-h”将出现中文帮助信息, 如果不带参数将会直接运行, 如果输入参

① 收稿时间:2015-11-10;收到修改稿时间:2016-03-17 [doi: 10.15888/j.cnki.csa.005398]

数不正确将会出现“命令行参数不正常,请参见:”的提示.

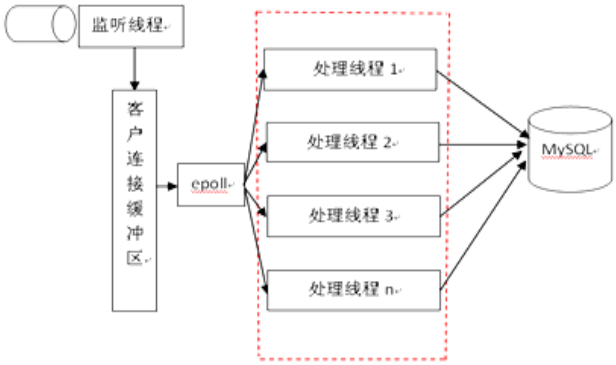

图 1 服务器任务处理框架

在上述就绪之后, 服务器将首先创建并初始化线程池, 创建网络环境并初始化参数. 接着调用epoll_create 函数来创建一个epollfd 的文件描述符. 第三, 调用epoll_ctl 函数将监听套接字注册到epoll事件的等待队列中. 第四是设计在一个“死循环”中, 调用调用epoll_wait 阻塞等待注册在epoll 上的sockfd事件的发生, 依次处理到来的事件, 如果到来的的事件是监听事件, 就调用accept 函数来产生连接套接字, 并且将这个套接字也设置成非阻塞的状态注册到epoll的事件等待的队列中, 如果到来的事件是表明有已经连接的客户端的通信数据到来, 就向线程池中抛入一个任务. 同时唤醒空闲的任务线程来处理相关的认证业务请求. 图2为高并发认证服务器流程图. 该设计最大的好处在于它不会随着监听fd数目的增长而降低效率. 因为在内核中的select实现中, 它是采用轮询来处理的, 轮询的fd数目越多, 自然耗时越多.

3 线程池技术

本系统采用线程池技术有效的降低了频繁创建销毁线程所带来的额外开销. 系统在服务器启动之初便预先创建一定数目的任务线程. 在运行的过程中, 需要时可以从这些线程所组成的线程池里申请分配一个空闲的线程, 来执行认证的任务, 任务完成后, 并不是将线程销毁, 而是将它返还给线程池. 如果线程池中预先分配的线程已经全部分配完毕, 但此时又有新的任务请求, 则线程池会动态的创建新的线程去适应这个请求. 以下将详细阐述本系统线程池设计[3,4].

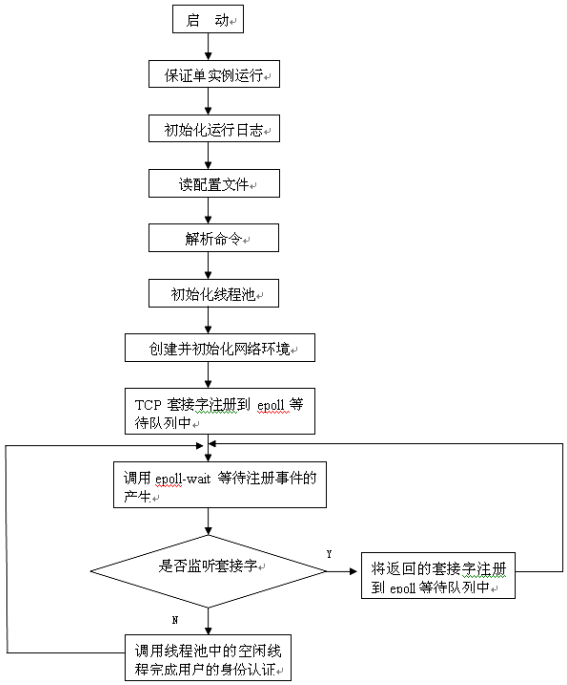

图 2 高并发认证服务器流程图

① 线程池设计

typedef struct tp_thread_pool_s TpThreadPool; //对线程池的描述.

struct tp_thread_pool_s

{ unsigned min_th_num; //线程池中至少存在的线程数, 线程池初始化的过程中会创建min_th_num数量的线程;

unsigned cur_th_num; //线程池当前存在的线程数量;

unsigned max_th_num; //线程池最多可以存在的线程数量;

pthread_mutex_t tp_lock; //用于线程池管理时的互斥;

pthread_t manage_thread_id; //线程池的管理线程ID;

TpThreadInfo* thread_info; //是指向线程池数据, 这里使用一个数组来存储线程池中线程的信息, 该数组的大小为

Queue idle_q;

TPBOOL stop_flag; //用于线程池的销毁, 当stop_flag 为 FALSE 时, 表明当前线程池需要销毁, 所有忙碌线程在处理完当前任务后会退出.

② 任务处理

当一个新任务到达时, 线程池首先会检查是否有可用的空闲线程, 如果是, 则采用才空闲线程进行任务处理并返回 TRUE, 如果不是, 则尝试新建一个线程, 并使用该线程对任务进行处理, 如果失败则返回 FALSE, 说明线程池忙碌或者出错. 以下函数是任务处理函数, 该函数将始终处理等待唤醒状态, 直到新任务到达或者线程销毁时被唤醒, 然后调用任务处理回调函数对任务进行处理; 当任务处理完成时, 则将自己置入空闲队列中, 以供下一个任务处理. 当线程池没有空闲线程可用时被调用.

函数将会新建一个线程, 并设置自己的状态为 busy(立即就要被用于执行任务).

```
staticvoid *tp_work_thread(void *arg)
{
pthread_t curid;
TpThreadInfo *pTinfo = (TpThreadInfo *) arg;
while (!(pTinfo->tp_pool->stop_flag))
{
pthread_mutex_lock(&pTinfo->thread_lock);
pthread_cond_wait(&pTinfo->thread_cond,&pTinfo->thr
ead_lock);
pthread_mutex_unlock(&pTinfo->thread_lock);
pTinfo->proc_fun(pTinfo->th_job);
pTinfo->is_busy = FALSE;
enQueue(&pTinfo->tp_pool->idle_q, pTinfo);
DEBUG("Job done, I am idle now.\n");
}
}
```

4　加密解密设计

高并发认证服务器的任务是服务器对客户端的身份做一个认证. 认证服务器首先生成一对公私钥对, 在接到客户端发来的认证请求后, 服务器将公钥发给客户端, 客户端接到服务器端的响应后, 生成 DES 密钥, 用接收来的服务器公钥加密用户帐号 UID,用户口令 USEPASSWORD,用户的 DES 密钥 DESKEY 等信息, 并将其发送给服务器, 服务器用私钥解密数据后, 验证用户身份的合法性.

对称加密 DES 算法把 64 位的明文输入块变为数据长度为 64 位的密文输出块, 其中 8 位为奇偶校验, 另外 56 位作为密码的长度. 首先, DES 把输入的 64 位数据块按位重新组合, 并把输出分为 L0、R0 两部分, 每部分各长 32 位, 并进行前后置换, 最终由 L0 输出左 32 位, R0 输出右 32 位, 根据这个法则经过 16 次迭代运算后, 得到 L16、R16, 将此作为输入, 进行与初始置换相反的逆置换, 即得到密文输出, 实际加密过程要分成两个同时进行的过程, 即加密过程和密钥生成过程. 图 3 为加密与密钥生成图. 在 16 轮循环的每一轮中, 密匙位移位, 然后再从密匙的 64 位中选出 48 位. 通过一个扩展置换将数据的右半部分扩展成 48 位, 并通过一个异或操作替代成新的 32 位数据, 再将其置换一次. 这四步运算构成中的函数 f. DES 算法的解密过程和加密过程几乎完全相同, 只是使用密钥的顺序相反. 验证认证服务器的加密解密算法客户端程序 数据结构如下:

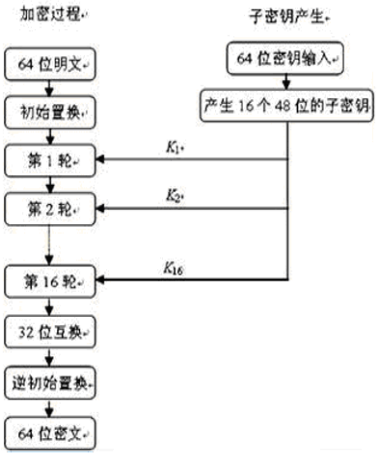

图 3　加密与密钥生成图

```
struct DES   //用例(客户端)的存储加密文件的结构体
{
    int a;          //服务器任务处理标记
    char uid[512]; //UID
    char deskeyszEncbuf[1024]; //deskey  加密密文
    char userpasswmiwen[1024]; //usepassword  加密密
文
}des;
struct MIMA   //密码结构体
{
    char mima1[512];   //usepassword 缓冲区
}mi;
struct PUBKEY
{
```

unsigned char pubkeystring[RSALEN]; //公钥字符串缓冲区
}key;

5 系统测试

5.1 系统运行环境

操作系统: Linux(ubuntu 10.10), Linux(内核 2.6 版本) .

编译环境: g++, openssl 库函数.

5.2 服务器端的安装配置

服务器端的安装配置包括服务器配置, 服务器端数据库配置. 在服务器配置中为了提高程序的可移植性, 用户可以根据自己的实际环境修改 conf 目录下的配置文件(server.conf)中相关变量的内容. 如 mysql 服务器的库名、用户名、密码, 服务器监听的端口号, 最大线程数和最小线程数等[5].

5.3 运行测试

系统集成测试设计了登录测试、 系统压力测试等. 登录测试为了验证认证服务器的加密解密算法的正确性, 设计了一个正常的客户端程序. 该程序首先生成 DES 密钥, 然后向服务器发送认证请求, 在接收到服务器的公钥后, 用公钥加密自己的用户 ID 号、口令和 DES 密钥发送给服务器. 测试的目的是查看服务器能否正确解密客户端信息并进行验证.

压力测试设计了发送错误的UID号与正确的UID号2个用例. 用例 1 是客户端不断发送错误的 UID 号(此次测试 UID=1234)进行一单进程的方式死循环访问服务器. 测试 5 分钟.从‘运行结果’和‘运行日志文件’进行分析, 由于 UID 错误, 结果只显示 connect success, 然后直接提示 uid error, 再关闭 SOCKET, 接着循环继续运行, 所以效果则是刷屏一样的不停的显示, 但不会因为压力太大而断开. 测试用例 2 是在客户端先发送正确的的 UID 号(此次测试 UID=12345)与密码(mima=123456789)进行不间断的死循环访问服务器测试 5 分钟, 从图 4、5 运行结果截图进行分析推论出:

一秒钟运行超过 300 次, 相当于对服务器访问了 300 次, 所以说服务器 1 秒钟能承受 300 次以上正确的 UID 与 mima 验证的访问压力. 5 分钟的访问压力也不会让服务器承受不了压力而断开或出错, 因此服务器承受正确的业务逻辑稳定.

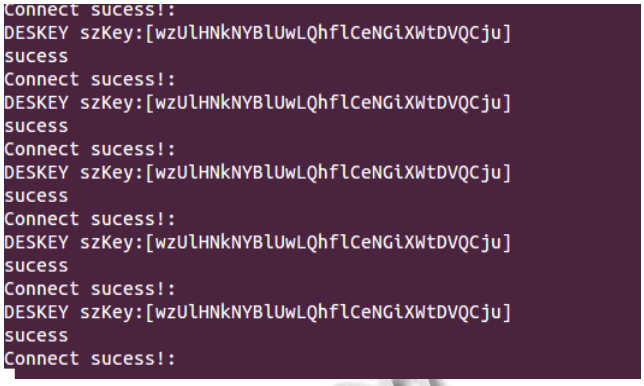

图 4 客户端循环发送正确 UID, 密码运行的截图

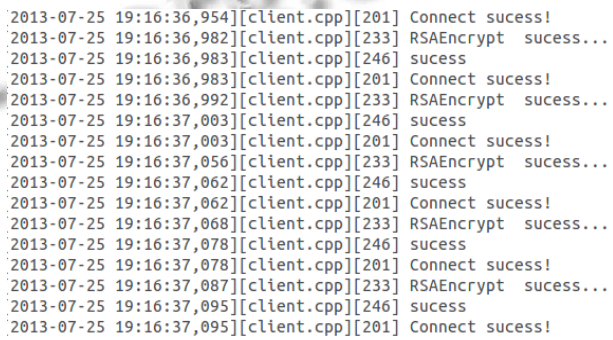

图 5 正确请求的压力测试日志效果截图

6 结语

本文设计并实现了一种高并发认证服务器基本功能, 具有并发、客户端与服务器双重认证的特点, 详细分析了采用 epoll 和线程池技术设计来保证其具有高并发功能, 分析采用了 RSA 和 DES 算法设计来保证信息传输的安全性, 通过完整设计实例测试表明, 认证服务器在高负载的情况下运转正常, 该系统作为在短时间内同时处理大量的来自客户端请求认证的研究具有一定的现实意义. 该系统还存在一些不足之处, 如提高系统监控功能, 故障报警机制等, 后续工作还有待完善.

参考文献

1 陈硕.Linux 多线程服务端编程.北京:电子工业出版社,2009.2.
2 熊齐,唐佳明.Linux 集群实时监控系统的一种设计方法,计算机系统应用,2013,22(9):50−53.
3 陈涛,任海兰.基于 Linux 的多线程池并发 Web 服务器设计.电子设计工程,2015,28(11):167−169.
4 樊扬轲,唐杰.一种高并发认证服务器的实现.电脑知识与技术,2015,11(24):47−49.
5 王志刚,江友华.MySQL 高效编程.北京:人民邮电出版社,2012.