维普资讯 http://www.cqvip.com

TCP Server 的动态线程 POOL 技术研究

A Study on the Technology of Dynamic Multithread POOL in TCP Server

赵荣彩

（中国科学院计算技术研究所　北京100080）

Abstract Multithread technique is one of the major technologies to implement parallel TCP Server. This paper presents a method of dynamic multithread POOL to enhance the performance of TCP Server, and simulates, analyzes and studies the several techniques for implementing dynamic multithread POOL. So, it gives us a reference to design a high performance parallel TCP Server.

Keywords TCP Server, Multithread, Thread POOL, Parallel programming

一、引言

在传统的 Unix 系统下，当一进程需要另一实体去完成某一子任务时，它就创建一个子进程进行处理。多年来，在 Unix 系统下的大多数支持并行访问的网络 Server 都是按这种模式设计和运行的，当父进程接受到来自网络的一个 Client 的连接时，它就 fork 一个子进程，并由该子进程去处理这个 Client 的请求。这种方法存在两个问题：

·fork 的开销比较大，要从父进程到子进程进行存储 copy 和复制父进程的各种描述等有关信息。并行实现中常采用 copy-on-write 优化技术来避免在 fork 时进行从父进程到子进程的立即数据复制，而是当子进程需要相应的数据时再进行复制。尽管如此，fork 的开销仍然不能明显地减小。

·在 fork 后，需要进程间通信（IPC）在父子进程间进行信息传递。在 fork 之前，这种信息传递当然是容易的，因为它是在 fork 时进行直接复制的。但是，从子进程往父进程返回时就有大量的工作要做。

线程（thread）技术可以很好地解决这两个问题。一般情况下，创建一个线程要比创建一个进程快10-100倍；一个进程中的所有线程可以共享相同的全局存储空间，使得线程之间的信息共享更为容易，但这样却带来了新的线程同步（synchronization）问题。一个进程中的所有线程间可共享的不仅仅是全局变量，还可共享以下信息：进程结构；大量的数据；打开的文件（文件描述符）；信号处理（handler）和信号分配；当前工作目录；用户和组标识（ID）。

而且，每个线程有它自己的：线程标识（ID）；包含程序计数器和堆栈指针的寄存器组；堆栈；错误码变量（errno）；信号屏蔽码和；优先级。

因此，近年来线程技术在并行处理和并发控制的开发和研究中开阔了新的研究领域和思想，并在技术和方法的研究上取得了长足的进展[1]。

在 Internet 迅速发展之今日，如何提高网络 TCP Server 的处理能力和性能是所有信息 Server 设计者不断追求的重要目标之一[2]。TCP Server 是一种典型的任务级并行处理系统，研究 TCP Server 设计中的线程技术和方法对提高并行处理的性能有一定的指导意义和代表性。本文在线程 POOL 技术的基础上进行了新的技术探讨、研究和分析，旨在为进一步挖掘 TCP Server 的处理能力和性能提供一点设计参考。

二、TCP Server 的线程 POOL 模型

在多线程并发 TCP Server 的设计和实现中，最基本的多线程并发模型是 Server 进程，首先运行的是主线程，主线程按照网络 API 编程要求，建立 TCP socket 并监听该端口，然后进入主循环过程，调用 accept，阻塞等待 client 连接的到来，一旦 accept 返回一个 client 的连接，马上就创建一个新的线程，由该新线程处理该 client 的服务请求，这时主线程再进入下一轮接收循环[3,4]。按照这种模式，虽然线程的创建比进程的创建要快得多，但仍然存在一定的延迟。如果采用线程 POOL 的思想，主线程在一开始就预先创建一定数量的线程，等待 client 连接请求的到来，一旦收到一个 client 的连接请求，就直接从线程 POOL 中给该线程分配一个线程为其服务，这样，对 client 的请求响应就快到几乎可以忽略延迟开销的速度。这种模式即为 TCP Server 线程 POOL 基本模型的思想。但是，在这种模式的具体设计实现中，存在两个关键的技术问题：调用 accept 阻塞等待的时机安排和线程 POOL 的调度与管理。这两个问题的合理解决对提高 TCP Server 的性能将会起到相当的作用。

1. 在子线程中调用 accept 的 TCP 线程 POOL Server 模型

调用 accept 等待 client 连接请求的时机和位置主要有两种选择：主线程调用和子线程调用。在主线程中调用 accept 时，由于每个 client 都是先与主线程建立完“三次握手”后再交由各子线程进行处理，因此，主线程调用 accept 模型没有子线程调用 accept 模型并行程度高。子线程调用 accept 的程序模型如图1所示，我们采用 POSIX Thread 标准编写。

```
#include "unpthread.h"
#include "pthread07.h"
// 线程结构
typedef struct {
  pthread_t thread_tid; //thread ID
  long thread_count // 处理的连接数
} Thread;
Thread *tptr; // 线程结构数组
int listenfd, nthreads;
socklen_t addrlen;
pthread_mutex_t mlock = PTHREAD_MUTEX_INITIALIZER;
int
main(int argc, char **argv)
{
  int i;
```

赵荣彩　研究员，从事多线程计算机体系结构研究，主要研究兴趣有计算机体系结构、操作系统和计算机网络。

维普资讯 http://www.cqvip.com

```
    void sig_int(int), thread_make(int);
    if (argc == 4)
        listenfd = Tcp_listen(NULL,argv[1], &addrlen);
    else if (argc == 5)
        listenfd = Tcp_listen(argv[1], argv[2], &addrlen);
    else
        err_quit("usage:serv07[ <host>]<port #> <#max-
        thread> <#inithread>");
    inithreads = atoi(argv[argc - 1]);
    maxthreads = atoi(argv[argc - 2]);
    tptr = Calloc(maxthreads, sizeof(Thread));
// 创建线程 POOL.
  for (i=0; i<inithreads; i++)
        thread_make(i);
  Signal(SIGINT, sig_int);
for ( ; ; )
     pause(); // 切换到 pool_manage().
}
void
thread_make(int i)
{
    void *thread_main(void *);
    pthread_create(&tptr[i], thread_tid, NULL, &thread_main,
    (void *)i );
    return;
}
void*
thread_main(void *arg)
{
    int connfd;
    void web_child(int);
    socklen_t clilen;
    struct sockaddr *cliaddr;
    cliaddr = Malloc( addrlen );
    for (;;){
        clilen = addrlen;
        pthread_mutex_lock(&mlock);
        connfd = Accept(listenfd, cliaddr, &clilen);
        pthread_mutex_unlock(&mlock);
    tptr[(int)arg].thread_count++;
    web_doit( connfd ); // 处理 client 的请求
    Close(connfd);
  }
}
```

图1　子线程调用 accept 的线程 POOL 程序模型

这是一种将连接分配到各个子线程去处理的并行技术。该模型的主线程根据命令行参数首先设置监听端口和预先创建的子线程数目；然后，分配各线程的数据结构空间，创建各线程和设置信号量；最后，进入循环等待，之后的所有工作全部交由各线程并发处理。

各子线程间通过对 accept 调用设置线程互斥信号量操作 pthread_mutex_lock 和 pthread_mutex_unlock 来控制各线程对 client 请求队列的读取。每个线程在读到一个 client 请求并建立连接后，调用 web_doit(connfd)去进行服务处理，处理完该请求后，关闭连接并循环进入下一轮的信号量检测和 accept 调用等待。各个线程就这样通过信号量 mlock 控制轮流读取 client 队列中的连接请求和进行并行处理[6]。

这一技术的优点是在处理 client 的请求时，没有创建子线程的时间开销；缺点是主线程必须把握要预先创建多少个子线程最合适，而且，一旦 POOL 中的线程被使用完，那么新到达的 client 就要等待直到 POOL 中有可用的线程时再处理到达的 client 连接请求。

2. 线程 POOL 的调整与管理

基于上述讨论，本文尝试采用一种对线程 POOL 进行动态管理的方法，根据 Server 当前的动态访问情况，实时调整线程 POOL 的大小。在图1中主线程的最后一个循环 pause() 处，将 pause 调用改为 pool_manage()，Server 不断监视 POOL 中可用线程的变化情况，一旦可用的线程数目少于了下限值 LowThreshold，Server 就再创建一定数量 increment 个可用线程；反之，若 POOL 中的可用线程多于了上限值 HighThreshold，Server 就终止一定数量 decrement 个线程，因为，太多的可用线程也会降低系统的性能，此现象亦称“异常群效应(the effect of the thundering herd)”。

三、线程 POOL 技术的性能分析与研究

本文在 Linux 环境下，对该模型进行了模拟和性能对比分析。在模拟的过程中，对 accept 调用分布、POOL 的管理方式、各相关参数的分布等进行了对比和分析。

1. 模拟环境的设定

硬件平台采用 IBM PC300，512M RAM，10M Ethernet。Client 模拟程序按均匀分布在1个小时之内通过10台 PC 随机发出[10..1000]个并发的访问请求，Server 中 web_doit()也按[1..100]秒的随机服务时间为 Client 进行服务。

2. 相关参数的设定

在我们的实验环境中，经过多次实验调整，我们设定命令行参数 maxthreads 为200，inithreads 为20，相对性能较佳。

3. POOL 管理模式对比

对 POOL 的管理模式主要体现在 pool_manage()函数的算法上，本文对三种不同算法进行了模拟。第一种是设定初始 POOL 的值 inithreads 为15，不再进行任何动态调整(基本模式)；第二种是动态观察 POOL 中可用线程的变化情况，若当前可用线程数低于 LowThreshold(=5)，就重新再创建20个线程，若当前可用线程数多于 HighThreshold(=35)，就使其可用线程数为15，多余的其他可用线程被终止；第三种是观察可用线程动态变化的速率，若 POOL 每10秒减少多于10个且可用线程又低于 LowThreshold(=5)，就重新再创建30个线程，否则，就重新再创建20个线程，若当前可用线程数多于 HighThreshold(=40)，就使其可用线程数为15，多余的其他可用线程被终止。经过对模拟数据进行统计平滑处理，图2中直方图依次分别给出了同时请求连接数在50，100，150，200时 Server 对 client 请求响应时间的对比图。

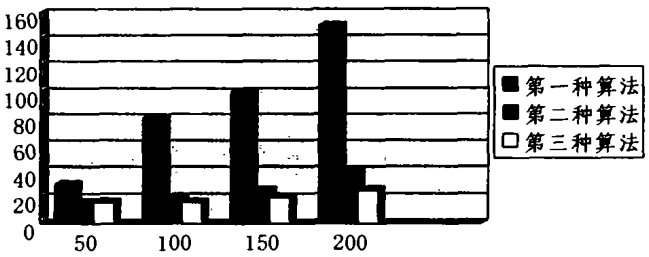

图2　各种情况下 Server 对 client 请求响应时间的对比

4. accept 调用的分布对比

图2中曲线1给出了将 accept 调用分别分布在主线程中时的情况对比，其它两种动态调整 POOL 的情况自然相对就会更好一些。

通过图2中的响应时间对比，可以看出不同技术间的性能还是有明显的区分的。在处理具体的统计数据时，还发现尤其是在同时 client 的请求数大于100以上时将 accept 调用分布在主线程要比在子线程中的最佳情况(同时请求连接数为200时)响应时间高数倍，最高可达7倍之多；按动态可用线程数变化速率调整比纯按当前可用线程数调整响应时间快约20%左右；静态线程 POOL 和两种动态线程 POOL 之间的响应时间区别也比较明显。

结束语　本文在模拟和分析过程中，选取的情况非常有

(下转第111页)

维普资讯 http://www.cqvip.com

所以比等长码13位要短些。熵值小，这意味着可用较短的码位来进行编码。

3.2 甲骨文字的编码效率

在甲骨文编码时，我们采用了直观的字母数字混合符，所以对甲骨文字编码的码元数大于2。现设甲骨文字符集有 n 个甲骨文字，使用 m 个码元对它进行编码，第 i 个甲骨文字的码长为 l_i，用 P_i 表示其根据汉字的使用频度推测的甲骨文字使用频度，根据 Huffman 最佳编码方法有[5]：

$$\log_m(1/P_i)<l_i<\log_m(1/P_i)+1 \quad (4)$$

其中 $\log_m(1/P_i)$ 是第 i 个甲骨文字的信息量。

由于一个甲骨文字的使用频度不一定刚好等于 m 的负整数幂，所以取上述不等式形式。下面用 P_i 乘以式(4)，并遍历对其求和得

$$\sum_{i=1}^{n}P_i\log_m(1/P_i)<\sum_{i=1}^{n}P_il_i<\sum_{i=1}^{n}[P_i\log_m(1/P_i)]+\sum_{i=1}^{n}P_i \quad (5)$$

根据 Huffman 编码公式 $r=L/H-1$，其最短平均码长为：$l_{min}=\sum_{i=1}^{n}P_il_i$。式中最后一项的概率和为1，而 $\sum_{i=1}^{n}P_i\log_m(1/P_i)$ 就是编码甲骨文字字符集以 m 为底的熵。如用 H_m 表示，则不等式(5)为：

$$H_m<l_{min}<H_m+1$$

其中的 H_m 就是 Huffman 最佳编码长度的理论下限，它指出任何文字编码方案的平均码长都不可能达到此下限。例如根据汉字的使用频度，在不考虑语言符号出现概率相互影响时，当汉语文本中的汉字容量 n≥12370时，得到 $H_2=-\sum_{i=1}^{n}P_i\log_2P_i=9.65$位[6]。

当编码码元数 m 增加时相应的最佳编码长度随之缩短。计算结果如表3所示。

表3

m	l_{min}	
	下限值	上限值
10	3	4
26	2.05	3.05
32	1.97	2.97
47	1.71	2.71

根据前面对甲骨文字根的模糊聚类分析，形成了32个码元，即 m=32。从表中可以看出，对甲骨文编码的平均码长大致接近于3，差不多处于最佳码长的上限左右。由于甲骨文字根有500多个，仅由32个码元表示，会出现同一码元的字根数较多，造成重码率较高的现象。按照最佳码长编码，试图获得一个编码效率高、重码率低的理想甲骨文编码是十分困难的。为了减小重码率，甲骨文编码采用增加编码规则，即在末码上按其结构类型取码，使得某些甲骨文编码的码长为4。即甲骨文编码的理论最佳码长为3，而实际编码时最多为4位码长。

结束语 本文吸取现代编码思想，运用模糊信息分析理论，建立了甲骨文信息处理的模糊信息模型。利用模型分析甲骨文的部件(字根)和码元的确定规则，使用32个字符(25个英文字母和7个阿拉伯数字)作为码元，与甲骨文中的500多个字根相对应，实现了一字一码的编码方案，首次对甲骨文进行了有效地编码。运用信息论中的熵理论，分析了这种编码的效率和科学性，得出甲骨文编码的最佳码长大致接近于3，从而为5000多个甲骨文字进行科学编码提供了理论基础。

在计算机甲骨文象形码输入法(含甲骨文字库)的基础上，进行象形逻辑模型和象形对象演算系统的研究；针对当前国内外关于汉字识别与密码学的研究现状，研究一种全新的古文字识别方法和现代网络环境下的加解密技术。这些都是需要进一步深入研究和探索的问题。

参 考 文 献

1 徐中舒. 甲骨文字典. 四川辞书出版社，1988
2 肖明，胡金柱，陈增照，舒忠梅，谭支鹏. 甲骨文象形码编码原理初探. 见：1998中文信息处理国际会议论文集，清华大学出版社，1998. 80～83
3 胡金柱. 模糊决策与决策支持. 华中师范大学出版社，1990
4 黄崇福，王家鼎. 模糊信息分析与应用. 北京师范大学出版社，1992
5 徐松，胡金柱. 甲象文象形码输入法的实现. 华中师大学学报(自然科学版)，1995，29(3)：299～302
6 陈一凡，胡宣华. 汉字键盘输入技术与理论基础. 清华大学出版社，广西科学技术出版社，1994
7 Corp A，Adobe Technical Note #5176：The Compact Font Formats Specification，1998，3
8 Janousek V，Tomas Vojnar State Spaces of Object-Oriented Petri Nets. In：Proc. of the Workshop on Concurrency within the MFCS/CSL'98 conf. organized，University of Brno，Czech，1998
9 Johnson R A，Hardgrave B C. Object-Oriented methods. current practices and attitudes，The Journal of Systems and Software，1999，48(1)：5～12

(上接第138页)

限，不过被选取的这几种情况具有典型性和一定的代表性，基本反映了具有统计意义的性能变化趋势和区别。更精细的调整有关参数或进一步改进 pool_manage()函数的算法，一定还会得到更好和更准确的分析结果；如果在多处理器系统上，多线程动态 POOL 技术将会显示出更优越的性能。

这里仅以 TCP Server 的设计为例，分析和研究了多线程动态 POOL 的设计技术，并且可以表明它对提高 Server 的性能有明显的作用。线程 POOL 的思想应该说来自于进程 POOL 的观念，所以，在进程 POOL 可以应用的很多地方都可以采用线程 POOL，并可减少开销和提高性能。不仅如此，在很多数据库 Server 和应用 Server 的设计中，尤其是具有多处理器或多机系统的环境，深入研究和应用动态线程 POOL 技术都将可能在不同程度上得到性能的提高。

参 考 文 献

1 MDMA - Multithreaded Daemon for Multimedia Access. http://www.ibiblio.org/mdma-release/mdma.html
2 An Approach for Constructing Web Enterprise Systems on Distributed Objects. http://www.redbooks.ibm.com/redpapers/pdfs/redp0025.pdf
3 Distributed Systems & IP Services. http://www.multithread.co.uk/
4 Multithread Programming Technology. http://www.npac.syr.edu/projects/pcrc/cpswt-e/toc/node1.htmlrg
5 Installed files for Apache Web Server，:www.linuxdoc.org
6 W Richard Stevens，UNIX networking Programming Volume I 2nd ed，networking APIs：Sockets and XTI，1998 by Printice Hall