Linux 下基于 TCP 的预先派生子进程服务器的 Socket 编程

和家强，刘彦隆
（太原理工大学 信息工程学院，山西 太原 030024）

摘要：描述了客户/服务器模型以及常见的服务器类型——基于 TCP 的并发服务器。在一个基于 TCP 回射服务器程序的基础上，结合实际 Web 应用中的多进程服务器模型，考虑到原有的客户/服务器交互存在的问题，改进了客户程序，设计了实现并发功能的客户程序、并发服务器以及预先派生子进程服务器。在装有 Linux 的 PC 上分别进行客户程序和服务器程序的测试。实验结果表明：在与并发客户的 TCP 交互中，与并发服务器相比，预先派生子进程服务器能够及时处理新的客户连接，且响应时间减小到并发服务器的三分之一，达到了对服务器性能优化的目的。

关键词：Socket；TCP；预先派生子进程；多进程
中图分类号：TP393.09 **文献标识码**：A **文章编号**：1674-6236(2011)03-0143-04

Socket programming of preforking server based on TCP in Linux

HE Jia-qiang, LIU Yan-long
(*College of Information Engineering, Taiyuan University of Technology, Taiyuan* 030024, *China*)

Abstract: This paper described the C/S model and the common server type-TCP-based concurrent server. In a TCP-based echo server program, combining with the practical model of multi-process server in Web applications, and considering the problems of original C/S interaction process, it improved the client program and designed the concurrent client, concurrent server and pre-forking server. Client and server programs ran on PCs with Linux respectively. The results showed that in the interaction with the TCP-based client, compared with the concurrent server, the preforking server processed new client connections timely and the response time of it reduced to one third of its counterpart, and this realized the optimization of the server's performance.

Key words: Socket; TCP; preforking; multi-process

近年来 Linux 操作系统的应用成为热点，而 Linux 作为一个开源的操作系统，因为其内核小、效率高、兼容性好和稳定性强等优点以及强大的网络服务功能，在网络应用中得到了极大的发展。在因特网的发展日益广泛和深入的今天，面对着用户的巨大访问量和复杂的网络环境，如何设计效率高、稳定性好、安全性强的客户/服务器成为了一个很有意义的议题。

1 客户/服务器模型

在设计客户/服务器程序时，必须在 2 种类型的交互中做出选择：无连接的风格或面向连接的风格[1]。这两种风格的交互直接对应于 TCP/IP 协议族所提供的 2 个主要的传输协议。如果客户和服务器使用用户数据报(UDP)进行通信，那么交互就是无连接的；如果使用传输控制协议(TCP)，则交互就是面向连接的。

面向连接风格的交互协议使编程更简单，程序更可靠，因此采用 TCP 协议的客户/服务器模型是我们的首选。

服务器可以分为 2 种类型：循环服务器(iterative server)和并发服务器(concurrent server)[2]。循环服务器描述在一个时刻只处理一个请求的一种服务器实现。并发服务器描述一个时刻可以处理多个请求的一种服务器。

本文讨论的是基于 TCP 协议的并发服务器。事实上，这是一种较常见的服务器类型，它适用于对每个请求进行少量处理，但是要求有可靠的传输。

2 基于 TCP 协议的 Socket 编程

TCP 协议中常用的 socket 类型共有 4 种[3]：字节流套接口(SOCK_STREAM)，数据报套接口(SOCK_DGRAM)，有序分组套接口(SOCK_SEQPACKET)和原始套接口(SOCK_RAW)。流式 socket 是一种面向连结的 socket，对应于面向连接的 TCP 服务应用。

字节流套接口的服务器进程和客户进程在通信前必须先建立连接，建立连接和通信的步骤如图 1 所示。

收稿日期：2010-09-08 **稿件编号**：201009015
作者简介：和家强(1984—)，男，江苏南京人，硕士研究生。研究方向：嵌入式系统，网络传输技术。

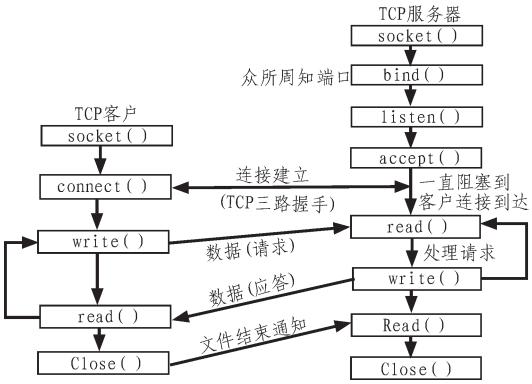

图 1　基于 TCP 的客户/服务器的传输流程图

Fig. 1　The flow diagram of TCP-based C/S transmission

1)服务进程首先调用 socket()创建一个字节流套接口，并调用 bind()将服务器地址捆扎在该套接口上，接着调用 listen()监听连接请求，随后调用 accept()做好与客户进程建立连接的准备，无连接请求时，服务进程被阻塞；

2)客户进程调用 socket()创建字节流套接口，然后调用 connect()向服务进程发出连接请求；

3)当连接请求到来后，服务进程被唤醒，生成一个新的字节流套接口，并用新套接口同客户进程的套接口建立连接，而服务进程最早生成的套接口则继续用于监听网络上的服务请求；

4)服务进程和客户进程通过调用 read()和 write()交换数据；

5)服务进程和客户进程通过调用 close()撤消套接口并中断连接。

3　客户/服务器的程序设计

3.1　对基于 TCP 的回射服务器的程序的改进

客户/服务器交互情形在 Web 应用中是典型的：客户向服务器发送一个小请求，服务器响应以返回给客户的数据。以一个回射服务器[4](如图 2 所示)为例，这是基于 TCP 协议的 Socket 编程，当从客户端输入文本时，可以从服务器返回刚才输入的文本。

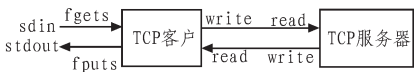

图 2　TCP 回射服务器

Fig. 2　TCP-based echo server

字节流套接口上的 read 和 write 函数所表现的行为不同于通常的文件 I/O。字节流套接口上的读或者写输入的字节数可能比要求的数量少，但这不是错误状况，原因是内核中套接口的缓冲区可能已达到了极限。为预防出现返回不足的字节计数值，设计了以下 3 个函数。当对字节流套接口进行读或写操作时，调用下面的 3 个函数：

ssize_t writen(int filedes, const void *buff,size_t nbytes);

writen 函数可以一次从 buff 写入 nbytes 个字节到文件描述符 filedes 中。

ssize_t readn(int filedes, const void *buff,size_t nbytes);

readn 函数可以一次从文件描述符 filedes 读入 nbytes 个字节到 buff 中。

ssize_t readline(int filedes, void *buff,size_t maxlen);

readline 函数可以一次从文件描述符 filedes 中读取 maxlen 字节的数据到 buff 中。

对于回射服务器程序存在两个问题。首先，进程在被阻塞以等待用户输入期间，看不到诸如对端关闭连接等网络事件。其次，它以停-等模式运作，批处理效率极低。对于这样的一个循环客户程序，可以通过调用 select 函数使得进程能够在等待用户输入期间得到网络事件通知。可以通过使用 shutdown 函数解决批处理效率低下的问题。

然而，实际的客户一般都不是循环客户。事实上，每个客户可能和若干个服务器发生若干次交互。本文从服务器的角度考虑，研究存在若干个客户，每个客户与一个服务器发生若干次的交互的情况。

3.2　客户算法

客户程序设计：每次运行本客户程序时，指定服务器的 IP 地址、服务器的端口、由客户 fork 的子进程数(以允许客户并发的向同一个服务器发起多个连接)nchildren、每个子进程发送给服务器的请求数 nloops 以及每个请求要求服务器返送的数据字节数 nbytes。

父进程调用 fork 派生指定个数的子进程，每个子进程再与服务器建立指定次数的连接。每次建立连接后，子进程就在该连接上向服务器发送一行文本，指出需要由服务器返回多少字节的数据，然后在该连接上读入这个数量的数据，最后关闭该连接。父进程只是调用 wait 等待所有子进程都终止。

TCP 客户调用子进程的程序如下：

```
for (i = 0; i < nchildren; i++) {
    if ( (pid = fork()) == 0) {          /* 子进程 */
        for (j = 0; j < nloops; j++) {
            fd = tcp_connect(argv[1], argv[2]);

            write(fd, request, strlen(request));

            if ((n = readn(fd, reply, nbytes)) ! = nbytes)
                err_quit("server returned %d bytes", n);

            close(fd);
        }
    printf("child %d done\n", i);
        exit(0);
    }
```

3.3　服务器算法

3.3.1　并发服务器

传统上并发服务器调用 fork 派生一个子进程来处理每

个客户。这使得服务器能够同时为多个客户服务,每个进程一个客户。并发服务器的问题在于为每个客户 fork 一个子进程比较耗费 CPU 时间。多进程并发服务器的流程如图 3 所示。

TCP 并发服务器的 main()函数调用子进程的程序如下:

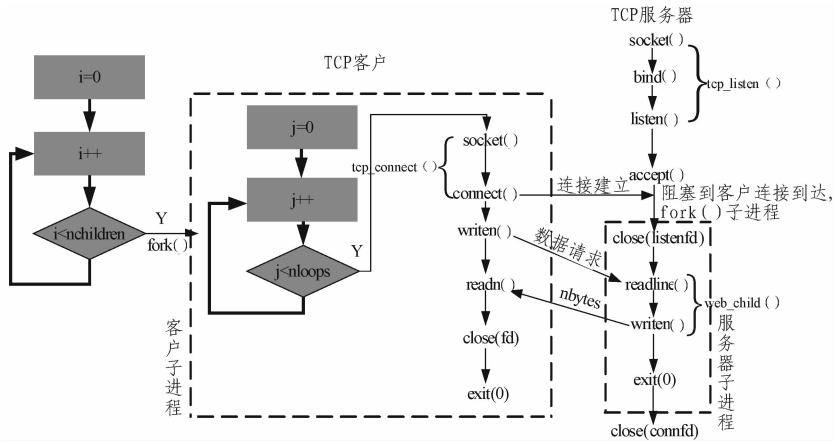

图 3　多进程并发服务器流程图

Fig. 3　The flow diagram of multi-process concurrent server

```
for ( ; ; ) {
        clilen = addrlen;
 if ( (connfd = accept(listenfd, cliaddr, &clilen)) < 0) {
            if (errno == EINTR)
                continue;
            else
                err_sys("accept error");
        }

        if ( (childpid = fork()) == 0) {/* 子进程 */
            close(listenfd);  /* 关闭监听套接口 */
web_child(connfd);    /* 处理请求 */
            exit(0);
        }
        close(connfd);  /* 父进程关闭套接口 */
 }
```

3.3.2　预先派生子进程服务器

预先派生子进程服务器是通过预先派生子进程来实现的。预先派生子进程[5]是让服务器在启动阶段调用 fork 创建一个子进程池。每个客户请求由当前可用子进程池中的某个(闲置)子进程处理(如图 4 所示)。

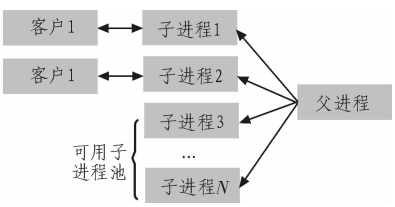

图 4　预先派生子进程服务器流程图

Fig. 4　The flow diagram of the preforking server

预先派生子进程服务器 main 函数的调用子程序的程序如下:

```
pids = calloc(nchildren, sizeof(pid_t));
for (i = 0; i < nchildren; i++)
    pids[i] = child_make(i, listenfd, addrlen);
```

派生各个子进程的 chli_make 函数:

```
pid_t child_make(int i, int listenfd, int addrlen)
```

每个子进程执行的无限循环的 clild_main 函数:

```
void child_main(int i, int listenfd, int addrlen)
```

3.4　程序的编译、测试和结果

1)在 Linux 上使用 gcc4.3.2[6]编译客户程序 client.c 和服务器程序 serv01.c 和 serv02.c,生成可执行文件 client(并发客户程序),serv01(并发服务器程序),serv02(预先派生子进程服务器程序),然后在一台装有 Linux 的 PC 上分别运行服务器程序的可执行文件 serv01 和 serv02。在另外两台装有 Linux 的 PC 上运行客户程序 client(如图 5 所示)。

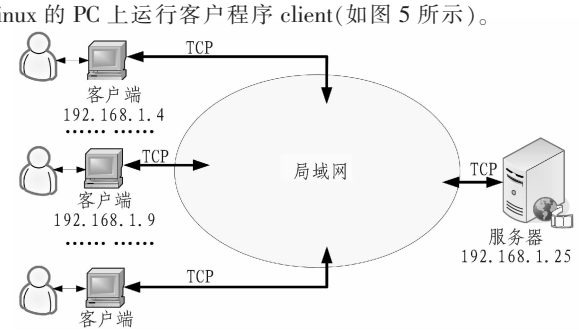

图 5　客户/服务器程序的性能测试

Fig. 5　The performance test of C/S programs

2)先在 PC 的终端上运行服务器程序:

./serv01 192.168.1.25 8888

(192.168.1.25 是服务器的 IP 地址,8888 是端口号)

然后分别在两台 PC 的终端上运行客户程序:

./client 192.168.1.25 8888 5 500 4000

(192.168.1.25 是服务器的 IP 地址,8888 是端口号,5 是子进程数,500 是每个子进程的 TCP 连接数,4 000 是返回的字节数)

当客户程序完成所有的 TCP 交互，在服务器的终端按“Crtl+C”中断,在终端得到用户 CPU 时间和系统 CPU 时间。

3)先在 PC 上运行服务器程序./serv02 192.168.1.25 8888 15

(192.168.1.25 是服务器的 IP 地址,8888 是端口号,15 是预先派生子进程数)

然后分别在两台 PC 上运行客户程序./client 192.168.1.25 8888 5 500 4000

当客户程序完成所有的 TCP 交互，在服务器的终端按“Crtl+C”中断,在终端得到用户 CPU 时间和系统 CPU 时间。

4)重复 3)步骤,设置预先派生子进程数为 30,45,60,75,90,105,400,1000 分别进行测试。

从表 1 可以看出预先派生子进程服务器的每次 TCP 交互的平均用户 CPU 时间和平均系统 CPU 时间都只有并发服务器的三分之一,服务器的响应时间得到了大大地提高。从表 2 可以得出这样的结论：随着预先派生子进程的增加,服务器的响应时间也在恶化。

表 1　预先派生子进程并发服务器和并发服务器的性能比较

Tab. 1　Performance comparison of preforking server and concurrent server

服务器	客户	用户 CPU 时间/秒	系统 CPU 时间/秒	CPU 时间/秒
	client(192.168.1.9)	0.076 004	3.592 22	
serv01		0.088 005	3.572 22	
(192.168.1.25)	client(192.168.1.4)	0.112 006	3.564 22	
		0.080 005	3.540 22	
	均值	0.089 005	3.567 22	3.656 225
serv02(192.168.1.25)	client(192.168.1.9)	0.016 001	1.088 07	
(192.168.1.25)		0.012 000	1.084 07	
(预先派生子进程数为 15)	client(192.168.1.4)	0.024 001	1.160 07	
		0.028 001	1.152 07	
	均值	0.020 001	1.121 07	1.141 071

表 2　预先派生子进程数对预先派生子进程服务器的性能的影响

Tab. 2　The impact of the number of preforking on preforking server's performance

服务器/客户	服务器的预先派生子进程数	用户 CPU 时间/秒	系统 CPU 时间/秒	CPU 时间/秒
	30	0.008	1.268 08	1.276 08
	45	0.016 001	1.208 07	1.224 071
serv02	60	0.036 002	1.144 07	1.180 072
(192.168.1.25)	75	0.028 001	1.220 08	1.248 081
client	90	0.016 001	1.240 08	1.256 081
(192.168.1.9)	105	0.020 001	1.244 08	1.264 081
	400	0.056 003	1.732 11	1.788 113
	1 000	0.023 601	2.420 15	2.443 751

4　结　论

本文在基于 TCP 的回射服务器的基础上，面对字节流 Socket 的缓冲区和 TCP 连接的交互的问题，改进了客户程序,实现了并发功能的客户以及并发服务器和预先派生子进程服务器。在与并发客户的 TCP 交互中,与传统的并发服务器相比较，预先派生子进程服务器能够及时的处理客户连接，并把服务器的响应时间降低到并发服务器的三分之一，提高了服务器的性能。但是服务器还不能总是应对客户负载的变动,当可用子进程数低于某个阈值时,会恶化服务器的响应时间。这需要设计相应的算法持续监视可用的子进程数。

(下转第 149 页)

表 1 紧急指针字节定义
Tab. 1 Emergency pointer bytes definition

位	优先权
第 0~2 位	优先权
第 3 位	0=普通延迟,1=低延迟
第 4 位	0=普通吞吐率,1=高吞吐率
第 5 位	0=一般可靠性,1=高可靠性
第 6~7 位	标志位

系列的操作使能将请求任务进行区分,设置优先级,服务器才能根据优先级对请求人物进行区别对待,使请求任务的要求的服务质量得到保证。

分类: 采用分类方法中对源 IP 地址识别分辨出数据包产生哪种应用,在 3 次握手中第一次握手当服务器接收到数据包时候, 根据用户的 IP 地址找到用户的相应信息判断用户的身份,区分用户是否是付费用户,在服务器中分为付费队列和非付费队列, 通过区分用户身份将其放入相应的队列,优先处理付费用户。然后解析分别对队列中数据包中的紧急指针,由此判断出请求任务要求的服务质量要求,即要求服务器需要提供服务的一种能力,包括专用带宽、抖动控制和延迟(用于实时和交互式流量情形)、丢包率的改进以及不同 WAN、LAN 和 MAN 技术下的指定网络流量等,同时确保为每种流量提供的优先权不会阻碍其它流量的进程。

进行标志位标注:为了能让服务器对分类结果能够区别对待,必须对分类结果进行标注,所以必须在数据包里面加入标注位,在分类结束之后根据分类结果对每个数据包进行标注,按照数据包中紧急指针显示出的服务质量要求区别进行标注,服务质量要求越高的标志位越高。

优先级设置:对数据包的标志位进行比较,根据标志位的大小,标志位越大优先级越高,依照此原则进行优先级设置。最后使得服务质量要求越高的请求任务优先级越高,服务器处理请求任务时候也能根据优先级的大小进行处理,真正做到区别对待。

2.5 处理请求任务

当将请求任务区分完之后, 需要对请求任务区分对待。根据前面的结果,需要服务器处理请求任务时候优先处理付费用户的队列,再优先处理其中高优先级的请求任务,当付费用户的请求任务处理完之后,再处理非付费用户的请求任务,同样也是优先处理其中高优先级的请求任务。

3 结束语

本文描述了如何让客户定义自己要求的服务质量和如何利用 Qos 对服务器的请求任务进行区分。做到进行区别对待,做了简单的设想。相比以前只有服务器资源得到比较有效利用,而客户的服务质量要求得不到表达,还是有客户的请求任务得不到满足,影响了客户对服务器的信息,造成了损失。当今服务器面临的客户请求任务越来越多,如果能做到对各种服务质量要求的请求任务进行区别对待,用户又能够自己描述自己服务质量,使服务器和客户都可以节省很多时间和资源,真正做到双赢。

参考文献:

[1] 林闯,单志广,盛立杰,等. Internet 区分服务及其几个热点问题的研究[J]. 计算机报,2000,23(4):419-433.
LIN Chuang, SHAN Zhi-guang, SHENG Li-jie, et al. Differentiated services in the Internet [J]. Chinese Journal of Computers, 2000,23(4)419-433.

[2] Alonso G, Casati F, Kuno H, et al. Web Service concepts [J]. Architectures and Applications,Berlin:Springer,2004.

[3] Ran S P. A model for Web services discovery with QoS[M]. ACM SIGecom Exchanges, 2003.

[4] Yu Jia. Buyya Rajkumar, Tham Chen Khong. QoS based scheduling of workflow applications on service grids [C]// Proceedings of the 1st IEEE international conference on e-Science and grid computing Melbourne, Australia, 2005.

[5] ITU2T. ITU2T2REC2X641—1997, Information technology quality of service : framework [S]. [S. l.]:ITU2T, 1997.

[6] DOGAN A, OZGUNER F. On Qos based scheduling of a meta-task with multiple Qos demands in heterogeneous computing [C]//Proc of Intl Parallel and Distributed Processing Symposium. Fort Lauderdale:[s. n.],2002.

(上接第 146 页)

参考文献:

[1] Comer D E. 用 TCP/IP 进行网际互联[M]. 1 卷.林瑶,张娟,王海,等,译. 北京:电子工业出版社,2007.

[2] Comer D E,Stevens D L. 用 TCP/IP 进行网际互联 [M]. 3 卷.赵刚,林瑶,蒋慧,等,译. 北京:电子工业出版社,2007.

[3] Stevens W R,Rago S A. Unix 环境高级编程[M]. 尤晋元,张亚英,戚正伟,译. 北京:人民邮电出版社,2006.

[4] RFC. 862. Echo Protocol[S]. 1983.

[5] Stevens W R,Rudoff A M. Unix 网络编程[M]. 1 卷. 杨继张,译. 北京:清华大学出版社,2006.

[6] 韦东山. 嵌入式 Linux 应用开发[M]. 北京:人民邮电出版社 2008.