针对**TCP**连接异常断开的分析
Posted on 2013-07-03 20:58 语行 阅读(8499) 评论(0) 编辑 收藏

我们知道，一个基于TCP/IP的客户端-服务器的程序中，正常情况下，我会是启动服务器使其在一个端口上监听请求，等待客户端的连接；通过T户端能够通过socket建立一个到服务器的连接；然后，两者就可以基于这个socket连接通信了。连接结束后，客户端（进程）会退出；在不需要继续处下，服务器（进程）也将退出。而且，当一个进程退出的时候，内核会关闭所有由这个进程打开的套接字，这里将触发TCP的四次挥手进而关闭一个so一些异常的情况下，譬如：服务器进程终止、服务器主机奔溃/奔溃后重启、服务器关机的情况下，客户端向服务器发起请求的时候，将会发生什么呢？这几种情况。

注意：一下描述的各种情况所使用的示例程序在文章的最后贴出

一、服务器进程终止

我们启动客户/服务器对，然后杀死子进程（模拟服务器进程崩溃的情形，我们可从中查看客户端将发生什么）。

1：在同一个主机上启动服务器和客户，并在客户上输入一行文本，以验证一切正常。正常情况下，改行文本将由服务器回射给客户。

2：找到服务器子进程的ID，通过kill命令杀死它。作为进程终止处理的部分工作，子进程中所有打开着的描述字都被关闭。这就导致向客户发送TCP则响应以一个ACK。这就是TCP连接终止的前一半工作。

3：子进程终止时，内核将给父进程递交SIGCHLD信号。

4：客户上没有发生任何特殊之事。客户TCP接受来自服务器TCP的FIN并响应一个ACK，然后问题是客户进程此时阻塞在fgets调用上，等待从终它是看不到这个FIN的。

5：此时我们如果运行netstat命令，可以看到如下的套接口的状态：

```
[ UNP1]$ netstat -nap | grep tcp | grep 9998
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 0.0.0.0:9998                0.0.0.0:*                   LISTEN      4222/
tcp        0      0 127.0.0.1:9998              127.0.0.1:50796             FIN_WAIT2   -
tcp        1      0 127.0.0.1:50796             127.0.0.1:9998              CLOSE_WAIT  4270/
[ UNP1]$
```

FIN_WAIT2即为我们杀掉的那个子进程的，因为我们知道主动关闭的那端在发送完fin并接受对端的ack后将进入fin_wait2状态，此时它在等待

6：现在我们在客户上在输入一行文本，我们可以看到如下的输出：

```
[ UNP1]$ ./client
test
test
after server close
server term prematurely.
```

当我们输入“after server close”时，客户TCP接着把数据发送给服务器，TCP允许这么做，因为客户TCP接受到FIN只是表示服务器进程已关闭而不再往其中发送任何数据而已。FIN的接受并没有告知客户TCP服务器进程已经终止（在这个例子中它缺失是终止了）。当服务器TCP接收到来自客户前打开那个套接口的进程已经终止，于是响应一个RST。

7：然而客户进程看不到这个RST，因为它在调用write后立即调用read，并且由于第2步中接收到FIN，所调用的read立即返回0（表示）EOF。未预期收到EOF，于是以出错信息“server term prematurely.”（服务器过早终止）退出。

8：当客户终止时，它所有打开着的描述字都被关闭。

我们的上述讨论还取决于程序的时序。客户调用*read*既可能发生在服务器的*RST*被客户收到之前，也可能发生在收到之后。如果*read*发生在收到子所示），那么结果是客户得到一个未预期的*EOF*；否则结果是由*readline*返回一个*ECONNRESET*（“*connection reset by peer*”对方复位连接）

本例子的问题在于：当FIN到达套接口时，客户正阻塞在fgets调用上。客户实际上在应对两个描述字——套接口和用户输入，它不能单纯阻塞在源的输入上，而是应该阻塞在其任何一个源的输入上。（可用select等io复用的函数实现）

二、服务器主机崩溃

我们接着查看当服务器主机崩溃时会发生什么。为了模拟这种情形，我们需要在不同的机器上运行客户与服务器，在首次确认客户服务器能正常上断开服务器主机，并在客户上再输入一行文本。这里同时也模拟了当客户发送数据时服务器主机不可达的情形（机建立连接后某些中间路由器不工作

1：当服务器主机崩溃时，已有的网络连接上发不出任何东西。这里我们假设的是主机崩溃，而不是执行了关机命令。

2：我们在客户上输入一行文本，它由write写入内核，再由客户TCP作为一个数据分节送出。客户随后阻塞于read调用，等待服务器的应答。

3：这种情况下，客户TCP持续重传数据分节，试图从服务器上接受一个ACK。（源自Berkeley的实现重传该数据分节12次，共等待约9分钟才TCP最终放弃时（假设这段时间内，服务器主机没有重新启动或者如果是服务器主机为崩溃但从网络上不可达的情况，那么假设主机仍然不可达），返误。既然客户阻塞在readline调用上，该调用将返回一个错误。假设服务器已崩溃，从而对客户的数据分节根本没有响应，那么所返回的错误是ETIME

个中间路由器判定服务器主机已不可达，从而响应以一个“destination unreachable”，那么所返回的错误是EHOSTUNREACH或ENETUNREACH。

尽管我们的客户最后还是发现对端主机已崩溃或不可达，不过有时候我们需要更快地检测出这种情况，而不是不得不等待9分钟。所用的方法就个超时。

另外我们刚讨论的情形只有在向服务器主机发送数据时，才能检测出它已经崩溃，如果我们不主动发送主句也想检测出服务器主机的崩溃，那么SO_KEEPALIVE这个套接口选项。

三、服务器主机崩溃后重启

在前一节的分析中，当我们发送数据时，服务器主机仍然处于崩溃状态；这节，我们将在发送数据前重新启动崩溃了的服务器主机。模拟这种情建立连接，再从网络上端口服务器主机，将它关机后再重启，最后把它重新连接到网络中。

如前一节所述，如果在服务器主机崩溃时客户不主动给服务器发送数据，那么客户不会知道服务器主机已经崩溃。所发生的步骤如下：

1：启动客户服务器，在客户上输入一行文本已确认连接已建立。

2：服务器主机崩溃并重启。

3：在客户上输入一行文本，它将作为一个TCP数据分节发送到服务器主机。

4：当服务器主机崩溃后重启时，它的TCP丢失了崩溃前的所有连接信息，因此服务器TCP对于所收到的来自客户的数据分节响应以一个RST。

5：当客户TCP收到该RST时，客户正阻塞于read调用，导致该调用返回ECONNRESET错误。

四、服务器主机关机

这节我们看看当服务器关机时将会发生什么。

Unix系统关机时，init进程通常先给所有进程发送SIGTERM信号（该信号可被捕获），再等待一段固定的时间（一般在5~20秒之间），然后给发送SIGKILL信号（该信号不能被捕获）。这么做是留给所有运行中的进程一小段时间来清除和终止。如果我们不捕获SIGTERM信号并终止，我们的信号终止。当服务器进程终止时，它的所有打开着的描述字都被关闭，随后发生的步骤与第一节中讨论过的一样。正如第一节中所述的情形，我们必须或poll函数，使得服务器进程的终止已经发生，客户马上检测到。

五、示例程序

```
//client.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/socket.h>
#include <errno.h>
#include <error.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <arpa/inet.h>
#include <string.h>
#include <signal.h>

void str_cli(FILE *fp, int sfd ) {
    char sendline[1024], recvline[2014];
    memset(recvline, 0, sizeof(sendline));
    memset(sendline, 0, sizeof(recvline));
    while( fgets(sendline, 1024, fp) != NULL ) {
        write(sfd, sendline, strlen(sendline));
        if( read(sfd, recvline, 1024) == 0 ) {
            printf("server term prematurely.\n");
        }
        fputs(recvline, stdout);
        memset(recvline, 0, sizeof(sendline));
        memset(sendline, 0, sizeof(recvline));
    }
}

int main() {
    int s;
    if( (s = socket(AF_INET, SOCK_STREAM, 0)) < 0 ) {
        int e = errno;
        perror("create socket fail.\n");
        exit(0);
    }

    struct sockaddr_in server_addr, child_addr;
    bzero(&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(9998);
    inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr);

    if( connect(s, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0 ) {
        perror("connect fail.");
        exit(0);
    }
    str_cli(stdin, s);
    exit(0);
```

```
}
```

```
//server.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/socket.h>
#include <errno.h>
#include <error.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <arpa/inet.h>
#include <string.h>
#include <signal.h>
#include <sys/wait.h>

//using namespace std;


typedef void sigfunc(int);

void func_wait(int signo) {
    pid_t pid;
    int stat;
    pid = wait(&stat);
    printf( "child %d exit\n", pid );
    return;
}
void func_waitpid(int signo) {
    pid_t pid;
    int stat;
    while( (pid = waitpid(-1, &stat, WNOHANG)) > 0 ) {
        printf( "child %d exit\n", pid );
    }
    return;
}

sigfunc* signal( int signo, sigfunc *func ) {
    struct sigaction act, oact;
    act.sa_handler = func;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    if ( signo == SIGALRM ) {
#ifdef            SA_INTERRUPT
        act.sa_flags |= SA_INTERRUPT;    /* SunOS 4.x */
#endif
    } else {
#ifdef           SA_RESTART
        act.sa_flags |= SA_RESTART;    /* SVR4, 4.4BSD */
#endif
    }
    if ( sigaction(signo, &act, &oact) < 0 ) {
        return SIG_ERR;
    }
    return oact.sa_handler;
}


void str_echo( int cfd ) {
    ssize_t n;
    char buf[1024];
    //char t[] = "SERVER ECHO: ";
again:
    memset(buf, 0, sizeof(buf));
    while( (n = read(cfd, buf, 1024)) > 0 ) {
        write(cfd, buf, n);
    }
    if( n <0 && errno == EINTR ) {
        goto again;
    } else {
        printf("str_echo: read error\n");
    }
}

int main() {

    signal(SIGCHLD, &func_waitpid);

    int s, c;
    pid_t child;
    if( (s = socket(AF_INET, SOCK_STREAM, 0)) < 0 ) {
        int e = errno;
        perror("create socket fail.\n");
```

```
            exit(0);
        }

        struct sockaddr_in server_addr, child_addr;
        bzero(&server_addr, sizeof(server_addr));
        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(9998);
        server_addr.sin_addr.s_addr = htonl(INADDR_ANY);

        if( bind(s, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0 ) {
            int e = errno;
            perror("bind address fail.\n");
            exit(0);
        }

        if( listen(s, 1024) < 0 ) {
            int e = errno;
            perror("listen fail.\n");
            exit(0);
        }
        while(1) {
            socklen_t chilen = sizeof(child_addr);
            if ( (c = accept(s, (struct sockaddr *)&child_addr, &chilen)) < 0 ) {
                perror("listen fail.");
                exit(0);
            }

            if( (child = fork()) == 0 ) {
                close(s);
                str_echo(c);
                exit(0);
            }
            close(c);
        }
    }
```

分类: 网络编程

语行
关注 - 0
粉丝 - 18
+加关注

0

« 上一篇：linux僵尸进程产生的原因以及如何避免产生僵尸进程
» 下一篇：套接字描述符就绪条件

刷新评论

最新知识库文章：
· 陈皓：什么是工程师文化？
· 没那么难，谈CSS的设计模式
· 程序猿媳妇儿注意事项
· 可是姑娘，你为什么要编程呢？
· 知其所以然（以算法学习为例）
» 更多知识库文章...