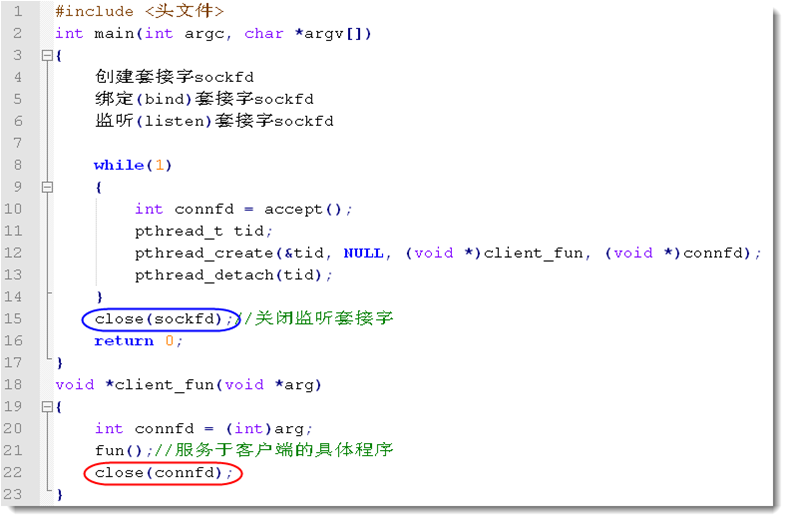
**tcp多线程并发服务器**

多线程服务器是对多进程服务器的改进，据统计，创建线程与创建进程要快 10100 倍，所以又把线程称为“轻量级”进程。

tcp多线程并发服务器框架:



我们在使用多线程并发服务器时，直接使用以上框架，我们仅仅修改client\_fun()里面的内容。

代码示例：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <pthread.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：   void \*client\_fun(void \*arg)

函数功能：   线程函数,处理客户信息

函数参数：   已连接套接字

函数返回：   无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void \*client\_fun(void \*arg)

{

    int recv\_len = 0;

    char recv\_buf[1024] = "";   // 接收缓冲区

int connfd = \*((int\*)arg); // 传过来的已连接套接字

    // 接收数据

    while((recv\_len = recv(connfd, recv\_buf, sizeof(recv\_buf), 0)) > 0)

    {

        printf("recv\_buf: %s\n", recv\_buf); // 打印数据

        send(connfd, recv\_buf, recv\_len, 0); // 给客户端回数据

    }

    printf("client closed!\n");

    close(connfd);  //关闭已连接套接字

    return  NULL;

}

//===============================================================

// 语法格式：    void main(void)

// 实现功能：    主函数，建立一个TCP并发服务器

// 入口参数：    无

// 出口参数：    无

//===============================================================

int main(int argc, char \*argv[])

{

    int sockfd = 0;             // 套接字

    int err\_log = 0;

    struct sockaddr\_in my\_addr; // 服务器地址结构体

    unsigned short port = 8080; // 监听端口

    pthread\_t thread\_id;

    printf("TCP Server Started at port %d!\n", port);

    sockfd = socket(AF\_INET, SOCK\_STREAM, 0);   // 创建TCP套接字

    if(sockfd < 0)

    {

        perror("socket error");

        exit(-1);

    }

    bzero(&my\_addr, sizeof(my\_addr));      // 初始化服务器地址

    my\_addr.sin\_family = AF\_INET;

    my\_addr.sin\_port   = htons(port);

    my\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    printf("Binding server to port %d\n", port);

    // 绑定

    err\_log = bind(sockfd, (struct sockaddr\*)&my\_addr, sizeof(my\_addr));

    if(err\_log != 0)

    {

        perror("bind");

        close(sockfd);

        exit(-1);

    }

    // 监听，套接字变被动

    err\_log = listen(sockfd, 10);

    if( err\_log != 0)

    {

        perror("listen");

        close(sockfd);

        exit(-1);

    }

    printf("Waiting client...\n");

    while(1)

    {

        char cli\_ip[INET\_ADDRSTRLEN] = "";     // 用于保存客户端IP地址

        struct sockaddr\_in client\_addr;        // 用于保存客户端地址

        socklen\_t cliaddr\_len = sizeof(client\_addr);   // 必须初始化!!!

        //获得一个已经建立的连接

        int connfd = accept(sockfd, (struct sockaddr\*)&client\_addr, &cliaddr\_len);

        if(connfd < 0)

        {

            perror("accept this time");

            continue;

        }

        // 打印客户端的 ip 和端口

        inet\_ntop(AF\_INET, &client\_addr.sin\_addr, cli\_ip, INET\_ADDRSTRLEN);

        printf("----------------------------------------------\n");

        printf("client ip=%s,port=%d\n", cli\_ip,ntohs(client\_addr.sin\_port));

        if(connfd > 0)

        {

            //由于同一个进程内的所有线程共享内存和变量，因此在传递参数时需作特殊处理，值传递。

            pthread\_create(&thread\_id, NULL,  client\_fun,  &connfd);  //创建线程

            pthread\_detach(thread\_id); // 线程分离，结束时自动回收资源

        }

    }

    close(sockfd);

    return 0;

}