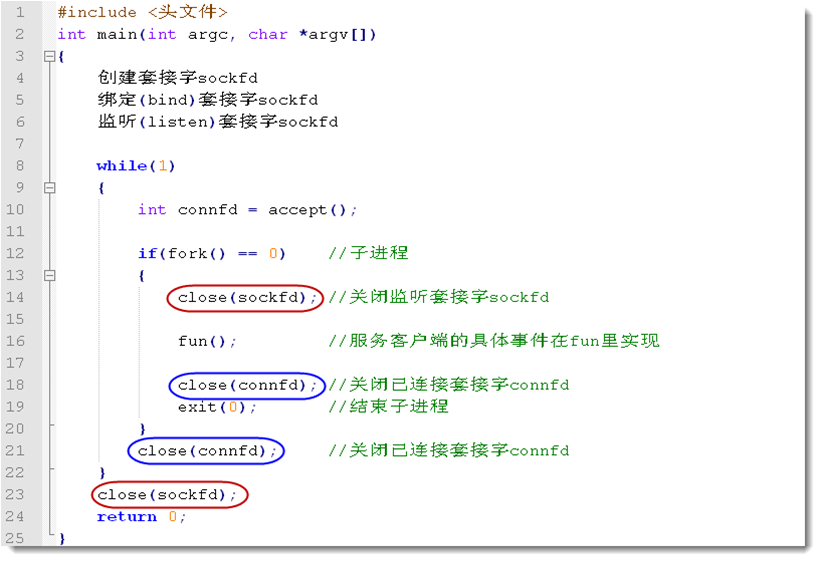
**一、tcp并发服务器概述**

一个好的服务器,一般都是并发服务器（同一时刻可以响应多个客户端的请求）。并发服务器设计技术一般有：多进程服务器、多线程服务器、I/O复用服务器等。

**二、多进程并发服务器**

多进程服务器是当客户有请求时，服务器用一个子进程来处理客户请求。父进程继续等待其它客户的请求。



tcp多进程并发服务器参考代码：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

int main(int argc, char \*argv[])

{

unsigned short port = 8080;     // 本地端口

    //1.创建tcp套接字

    int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

    if(sockfd < 0)

    {

        perror("socket");

        exit(-1);

    }

    //配置本地网络信息

    struct sockaddr\_in my\_addr;

    bzero(&my\_addr, sizeof(my\_addr));     // 清空

    my\_addr.sin\_family = AF\_INET;         // IPv4

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

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

    //2.绑定

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

    if( err\_log != 0)

    {

        perror("binding");

        close(sockfd);

        exit(-1);

    }

    //3.监听，套接字变被动

    err\_log = listen(sockfd, 10);

    if(err\_log != 0)

    {

        perror("listen");

        close(sockfd);

        exit(-1);

    }

    while(1) //主进程 循环等待客户端的连接

    {

        char cli\_ip[INET\_ADDRSTRLEN] = {0};

        struct sockaddr\_in client\_addr;

        socklen\_t cliaddr\_len = sizeof(client\_addr);

        // 取出客户端已完成的连接

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

        if(connfd < 0)

        {

            perror("accept");

            close(sockfd);

            exit(-1);

        }

        // fork（）函数通过系统调用创建一个与原来进程几乎完全相同的进程

        // 1）在父进程中，fork返回新创建子进程的进程ID；

        // 2）在子进程中，fork返回0；

        // 3）如果出现错误，fork返回一个负值；

        pid\_t pid = fork();

        if(pid < 0){

            perror("fork");

            \_exit(-1);

        }

        else if(0 == pid){ // 在子进程中pid为0

            // 接收客户端的信息，并发还给客户端

            /\*关闭不需要的套接字可节省系统资源，

              同时可避免父子进程共享这些套接字

              可能带来的不可预计的后果

            \*/

            close(sockfd);   // 关闭监听套接字，这个套接字是从父进程继承过来

            char recv\_buf[1024] = {0};

            int recv\_len = 0;

            // 打印客户端的 ip 和端口

            memset(cli\_ip, 0, sizeof(cli\_ip)); // 清空

            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));

            // 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\_port %d closed!\n", ntohs(client\_addr.sin\_port));

            close(connfd);    //关闭已连接套接字

            exit(0);

        }

        else if(pid > 0){   // 在父进程中pid大于0

            close(connfd);    //关闭已连接套接字

        }

    }

    close(sockfd);

    return 0;

}