****

**计算机网络实验报告**

Socket编程基础



**学 院 国际工程学院**

**年 级 2018**

**专 业 计算机技术**

**学 号 2018229038**

**姓 名 李亚蓉**

**2018年 11月29 日**

**Socket编程基础**

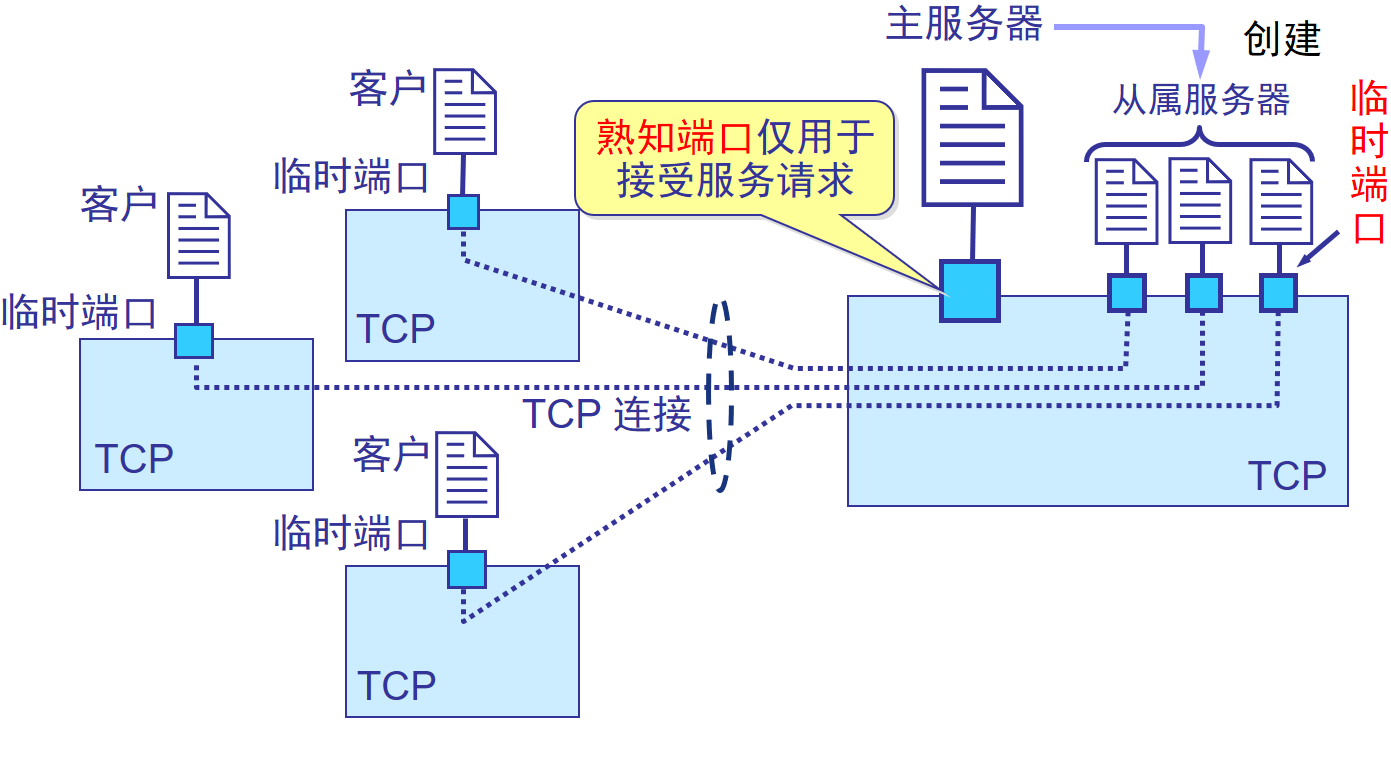
1. **实验目的**

基于多线程/进程机制使用Socket编写 能同时接纳多个 TCP的网络服务程序，实现服务器能同时和多个客户端并发通信的功能

1. **实验原理**

服务端的并发性：

并发性是C/S模式的基础，并发允许多个客户获得同一种服务，而不必等待服务器完成对上一个请求的处理。这样才能很好地同时为多个客户提供服务。



1. 基于多进程：

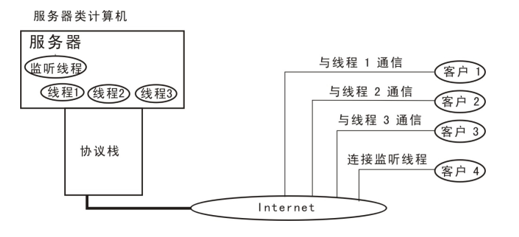
调用fork()函数，创建子进程，将所有的客户端的请求处理的内容都放在子进程中处理。

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。这种方法的优点是当客户有请求时，服务器能及时处理客户，特别是在客户服务器交互系统中。对于一个 TCP 服务器,客户与服务器的连接可能并不马上关闭，可能会等到客户提交某些数据后再关闭，这段时间服务器端的进程会阻塞，所以这时操作系统可能调度其它客户服务进程，这比起循环服务器大大提高了服务性能。

1. 基于多线程：

每一个客户端都分配给了一个fd值，即每一个客户端的线程都有各自的socket连接套接字。 在服务器程序中，主程序中仅定义了一个线程标识符，而且仅创建了一个线程，这样就可以进行多个客户端的连接。由于 socket的accept在while循环中，所以每有一个客户端请求连接服务器，都会生成一个新的连接套接字connectfd，而多个客户端连接上服务器后，共享一个线程的内存，各个客户端的线程之间并不是真正的并发、并行，而是线程的转换速度非常快，不断在多个客户端之间切换。这样以来，就可以看做多个线程并行地和服务器进行通信。

1. **实验过程**



服务端：

1.创建tcp套接字 ； 2. 绑定套接字 ； 3. 监听套接字 ； 4. 调用accept()阻塞等待 ；

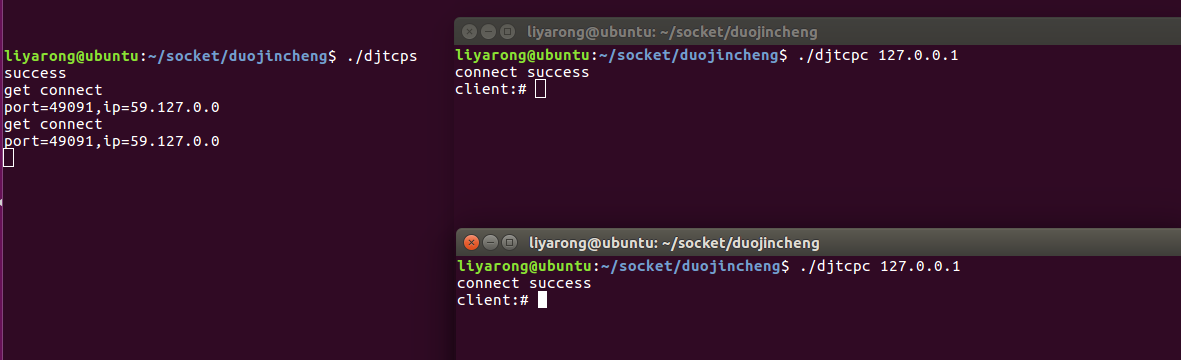
5. 处理客户端的请求，基于多进程的是通过调用fork()函数创建子进程来处理；基于多线程是通过创建一个线程来进行多个客户端的连接；

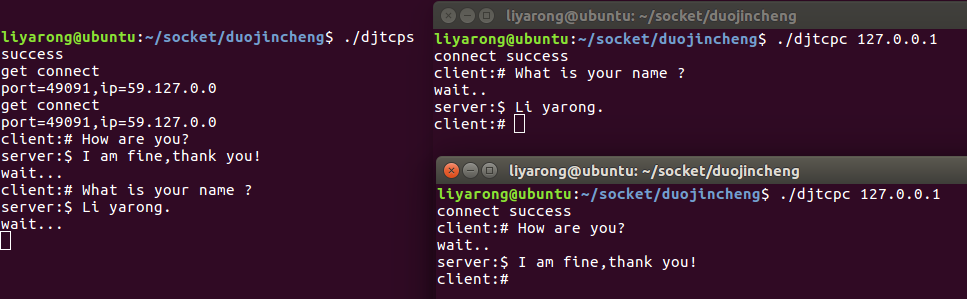
6. 关闭连接套接字； 7. 关闭监听套接字；

客户端：

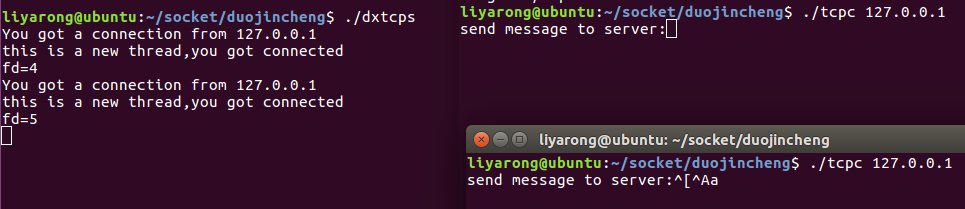
1.创建tcp套接字; 2.调用connect()连接服务器 ; 3.处理服务器端返回的信息

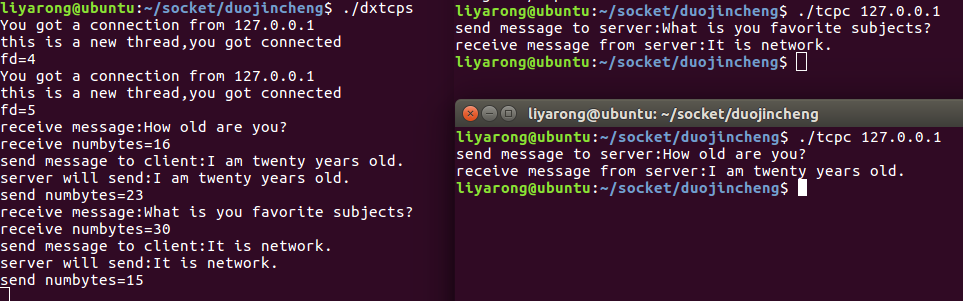
1. **实验结果**
2. 基于多进程





1. 基于多线程





1. **体会**

通过本次实验，使用Socket编写了基于多线程/进程机制，能同时接纳多个 TCP的网络服务程序，实现服务器能同时和多个客户端并发通信的功能，对Socket有了更进一步的了解，同时也增强了自己的实践能力，在阅读理解代码的基础上，对其进行了修改，唯一做的不足的地方是代码有点过于死板，应该要做到一个客户端可以对应多个服务端，希望下次再接再厉，感谢实验过程中老师和同学的帮助。

1. **源代码**

1、基于多进程

（1）客户端

#include<stdio.h>

#include<unistd.h>

#include<sys/socket.h>

#include<string.h>

#include<errno.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<sys/types.h>

#define SERVER\_PORT 9999

int main(int argc,char\* argv[])

{

if(argc!=2)

{

printf("Usage:client IP\n");

return 1;

}

char \*str=argv[1];

char buf[1024];

memset(buf,'\0',sizeof(buf));

struct sockaddr\_in server\_sock;

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

bzero(&server\_sock,sizeof(server\_sock));

server\_sock.sin\_family=AF\_INET;

inet\_pton(AF\_INET,str,&server\_sock.sin\_addr);

server\_sock.sin\_port=htons(SERVER\_PORT);

int ret=connect(sock,(struct sockaddr \*)&server\_sock,sizeof(server\_sock));

if(ret<0)

{

printf("connect()\n");

return 1;

}

printf("connect success\n");

while(1)

{

printf("client:# ");

fgets(buf,sizeof(buf),stdin);

buf[strlen(buf)-1]='\0';

write(sock,buf,sizeof(buf));

if(strncasecmp(buf,"quit",4)==0)

{

printf("quit\n");

break;

}

printf("wait..\n");

read(sock,buf,sizeof(buf));

printf("server:$ %s\n",buf);

}

close(sock);

return 0;

}

（2）服务端

#include<stdio.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<errno.h>

#include<unistd.h>

#include<string.h>

#include<sys/types.h>

#include<arpa/inet.h>

#include<netinet/in.h>

#define \_PORT\_ 9999

#define \_BACKLOG\_ 10

int main()

{

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

if(sock<0)

{

printf("socket()\n");

}

struct sockaddr\_in server\_socket;

struct sockaddr\_in socket;

bzero(&server\_socket,sizeof(server\_socket));

server\_socket.sin\_family=AF\_INET;

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

server\_socket.sin\_port=htons(\_PORT\_);

if(bind(sock,(struct sockaddr\*)&server\_socket,sizeof(struct sockaddr\_in))<0)

{

printf("bind()\n");

close(sock);

return 1;

}

if(listen(sock,\_BACKLOG\_)<0)

{

printf("listen()\n");

close(sock);

return 2;

}

printf("success\n");

for(;;)

{

socklen\_t len=0;

int client\_sock=accept(sock,(struct sockaddr\*)&socket,&len);

if(client\_sock<0)

{

printf("accept()\n");

return 3;

}

char buf\_ip[INET\_ADDRSTRLEN];

memset(buf\_ip,'\0',sizeof(buf\_ip));

inet\_ntop(AF\_INET,&socket.sin\_addr,buf\_ip,sizeof(buf\_ip));

printf("get connect\n");

pid\_t fd=fork();

if(fd<0)

printf("fork()\n");

if(fd==0)

{

close(sock);//关闭监听套接字

printf("port=%d,ip=%s\n",ntohs(socket.sin\_port),buf\_ip);

while(1)

{

char buf[1024];

memset(buf,'\0',sizeof(buf));

read(client\_sock,buf,sizeof(buf));

printf("client:# %s\n",buf);

printf("server:$ ");

memset(buf,'\0',sizeof(buf));

fgets(buf,sizeof(buf),stdin);

buf[strlen(buf)-1]='\0';

if(strncasecmp(buf,"quit",4)==0)

{

printf("quit\n");

break;

}

write(client\_sock,buf,strlen(buf)+1);

printf("wait...\n");

}

close(fd);

}

else if(fd>0)

{

close(fd);

}

}

close(sock);

return 0;

}

2、基于多线程

（1）客户端

#include <stdio.h>

#include <unistd.h>

#include <strings.h>

#include<string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <netdb.h>

#define PORT 1234

#define MAXDATASIZE 100

char receiveM[100];

char sendM[100];

int main(int argc, char \*argv[])

{

int fd, numbytes;

struct hostent \*he;

struct sockaddr\_in server;

//检查用户输入，如果用户输入不正确，提示用户正确的输入方法

if (argc !=2) { printf("Usage: %s <IP Address>\n",argv[0]);

exit(1);

}

// 通过函数 gethostbyname()获得字符串形式的ip地址，并赋给he

if ((he=gethostbyname(argv[1]))==NULL){

printf("gethostbyname() error\n");

exit(1);

}

// 产生套接字fd

if ((fd=socket(AF\_INET, SOCK\_STREAM, 0))==-1){

printf("socket() error\n");

exit(1);

}

bzero(&server,sizeof(server));

server.sin\_family = AF\_INET;

server.sin\_port = htons(PORT);

server.sin\_addr = \*((struct in\_addr \*)he->h\_addr);

if(connect(fd, (struct sockaddr \*)&server,sizeof(struct sockaddr))==-1){

printf("connect() error\n");

exit(1);

}

// 向服务器发送数据

printf("send message to server:");

fgets(sendM,100,stdin);

int send\_le;

send\_le=strlen(sendM);

sendM[send\_le-1]='\0';

send(fd,sendM,strlen(sendM),0);

// 从服务器接收数据

if ((numbytes=recv(fd,receiveM,MAXDATASIZE,0)) == -1){

printf("recv() error\n");

exit(1);

}

printf("receive message from server:%s\n",receiveM);

close(fd);

}

（2）服务端

#include <stdio.h>

#include <string.h>

#include <strings.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include<pthread.h>

#define PORT 1234

#define BACKLOG 1

void \*start\_routine( void \*ptr)

{

int fd = \*(int \*)ptr;

char buf[100];

int numbytes;

int i,c=0;

printf("this is a new thread,you got connected\n");

printf("fd=%d\n",fd);

if ((numbytes=recv(fd,buf,100,0)) == -1){

printf("recv() error\n");

exit(1);

}

printf("receive message:%s\n",buf);

printf("receive numbytes=%d\n",numbytes);

//char str[numbytes];

char buffer[100];

printf("send message to client:")

fgets(buffer,100,stdin);

int send\_le;

send\_le=strlen(buffer);

buffer[send\_le-1]='\0';

//numbytes=send(fd,buffer,sizeof(buffer),0);

printf("send numbytes=%d\n",send\_le);

close(fd);

}

int main()

{

int listenfd, connectfd;

struct sockaddr\_in server;

struct sockaddr\_in client;

int sin\_size;

sin\_size=sizeof(struct sockaddr\_in);

pthread\_t thread; //定义一个线程号

if ((listenfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1)

{

perror("Creating socket failed.");

exit(1);

}

int opt = SO\_REUSEADDR;

setsockopt(listenfd, SOL\_SOCKET, SO\_REUSEADDR, &opt, sizeof(opt));

bzero(&server,sizeof(server));

server.sin\_family=AF\_INET;

server.sin\_port=htons(PORT);

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

// 绑定

if (bind(listenfd, (struct sockaddr \*)&server, sizeof(struct sockaddr)) == -1) {

perror("Bind error.");

exit(1);

}

// 监听

if(listen(listenfd,BACKLOG) == -1){ /\* calls listen() \*/

perror("listen() error\n");

exit(1);

}

while(1)

{

// accept

if ((connectfd = accept(listenfd,(struct sockaddr \*)&client,&sin\_size))==-1) {

perror("accept() error\n");

exit(1);

}

printf("You got a connection from %s\n",inet\_ntoa(client.sin\_addr) );

pthread\_create(&thread,NULL,start\_routine,(void \*)&connectfd);

}

close(listenfd);

}