# Linux平台的Socket基础应用-TCP通信程序

实验名称：Linux平台的Socket基础应用-TCP通信程序

学时安排：2课时 指导老师：李赞

实验类别：验证型、设计型 实验要求：1人1组

学号： 姓名： 班级：

## 一、实验目的和任务

1. 本实验要求复习ubuntu的IP配置知识。

2. 掌握虚拟机桥接设置方法。

3. 了解使用socket编程的TCP流程。

## 二、实验设备介绍

1.软件需求： win10操作系统，VMware workstation，ubuntu18。

2.硬件需求: PC内存大于1G，硬盘空间大于20G。

## 三、注意事项和要求

1.启动虚拟机前应首先使用"虚拟网络编辑器"配置要桥接的网卡。

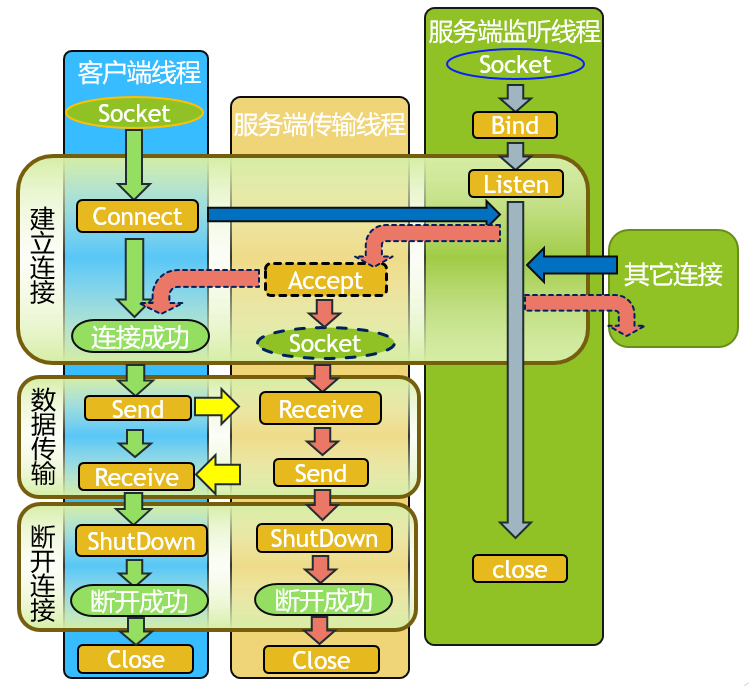
2.区分TCP服务端监听状态的socket对象与通信状态的socket对象。

3.调试程序时需运行两个程序联网调试。

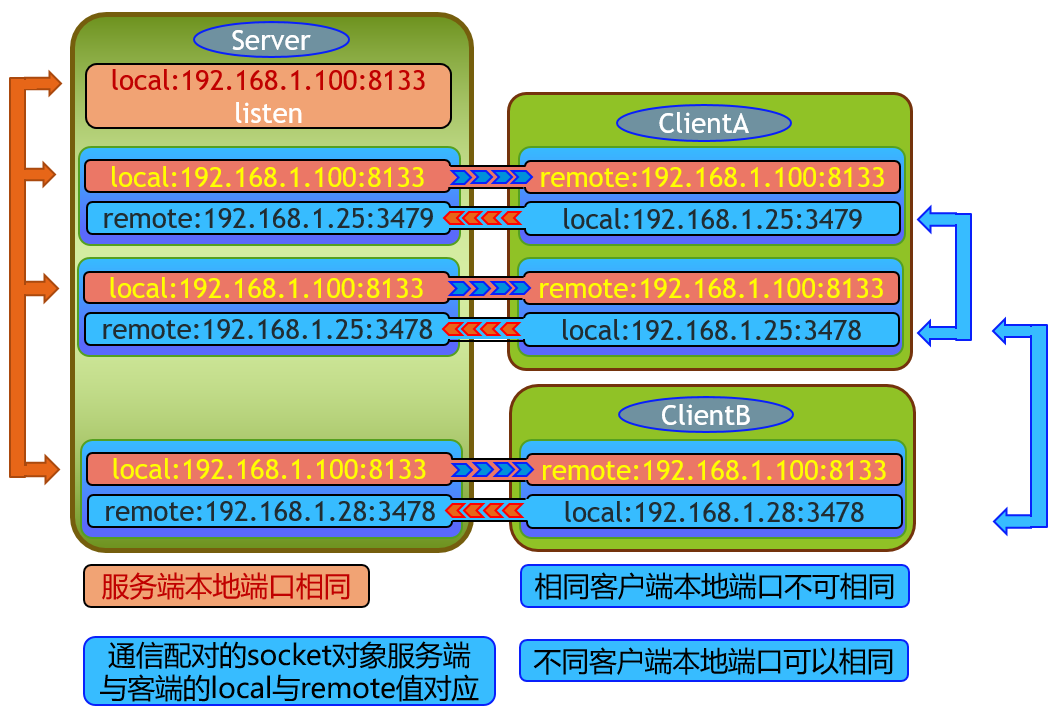
## 四、实验内容和步骤

### 4.1简单的TCP服务端与客户端通讯

TCP通讯模式如下图所示，服务端具有负责监听的socket，当客户端有连接请求时，服务端创建一个新的socket对象，与客户端socket形成配对关系，然后执行数据的相互发送，当双方需要断开时，通过调用shutdown方法实现稳妥断开。TCP是网络中最常用的面向连接的网络层协议，HTTP、FTP等协议就是基于TCP协议的应用层协议。



TCP通讯模式



TCP中服务端与客户端的Socket资源配对关系

下面采用TCP通讯的客户端hello\_client.c与服务端hello\_server.c程序实现一次网络信息的发送和接收，请仔细阅读程序，理解TCP通读中socket各方法的使用。TCP使用bind方法时，指定的绑定IP地址为本机IP，如果赋值为[INADDR\_ANY](https://www.baidu.com/s?wd=INADDR_ANY&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBrycLmHcknyn3mWFWnWfL0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1fzrHbsnHRs" \t "_blank)，那么系统将绑定默认的网卡【默认的IP地址】

#### 4.1.1客户端代码hello\_client.c

TCP客户端主要使用的方法有socket,connect,send,receive,shutdown,close。

//025.c TCP客户端hello程序

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

void error\_handling(char \* message);

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

{

int sock;

struct sockaddr\_in serv\_addr;

char message[30];

int str\_len;

//命令 IP PORT

if(argc!=3)

{

printf("Usage :%s <IP> <PORT>\n",argv[0]);

exit(1);

}

// 创建TCP套接字

//准备服务器的地址

// 对服务器发起连接请求

// 接收服务器发回的消息并打印

close(sock);

return 0;

}

//错误处理是将信息输出到标准输出，然后退出进程。

void error\_handling(char \* message)

{

fputs(message,stderr);

fputc('\n',stderr);

exit(1);

}

#### 4.1.2服务端代码hello\_server.c

//TCP服务端主要使用的方法有socket,bind,listen,accept,send,receive,shutdown,close。

//026.c TCP服务端hello程序

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

void error\_handling(char \* message);

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

{

int serv\_listen\_sock;//服务端负责监听的socket文件标识

int serv\_data\_sock;//服务端负责数据通信的socket文件标识

struct sockaddr\_in serv\_addr;

struct sockaddr\_in clnt\_addr;

socklen\_t clnt\_addr\_size;

char message[]="Hello embedded.";

if(argc!=2)

{

printf("Usage:%s <port>\n",argv[0]);

exit(1);

}

//创建一个TCP的socket资源

serv\_listen\_sock=socket(PF\_INET,SOCK\_STREAM,0);

if(serv\_listen\_sock==-1)

error\_handling("serv\_listen\_sock create() error");

memset(&serv\_addr,0,sizeof(serv\_addr));

serv\_addr.sin\_family=AF\_INET;

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

serv\_addr.sin\_port=htons(atoi(argv[1]));

//端口绑定

//执行监听

//等待客户端连接，实现连接请求响应

write(serv\_data\_sock,message,sizeof(message));

shutdown(serv\_data\_sock,SHUT\_RDWR);

close(serv\_data\_sock);

close(serv\_listen\_sock);

return 0;

}

//错误处理是将信息输出到标准输出，然后退出进程。

void error\_handling(char \* message)

{

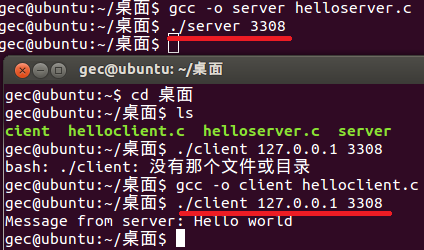
fputs(message,stderr);

fputc('\n',stderr);

exit(1);

}

在ubuntu环境的运行结果。



在机房的两台不同机器设置好IP，使得两个程序在两台机上完成通信。

在程序中添加下面的参考代码，实现双方聊天功能。

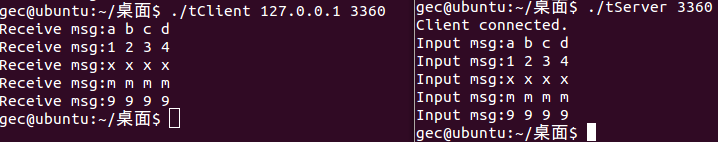
memset(message,0,30);

scanf("%[^\n]%\*c",message);

send(serv\_data\_sock,message,sizeof(message),0);

用户的输入数据首先被暂存在临时缓冲区中，当用户键入回车键或临时缓冲区满后，stdin 才进行 I/O 操作，将数据由临时缓冲区拷贝至 stdin 中。C语言提供的输入输出函数如 scanf 、getchar 等则从上述缓冲区 stdin 中按照规则读取数据，但不一定取走所有输入数据，当缓冲区数据为空时，**scanf 等函数再次等待用户输入。**用scanf()函数输入一个带空格的字符串函数时，经常会在第一个空格时就停止输入，使用%[^\n]就可以一直读入，直到碰到'\n'结束读入。可使用下面的方法输入包含空格的字符串，并且不影响后续的输入解析。

scanf("%[^\n]%\*c",message);



### 4.2 TCP服务端返回当前时间

本项目由客户端和服务端组成，客户端向服务端发送请求，服务端返回服务器的时间值，客户端收到信息后显示时间信息。

#### 4.2.1获得本机时间

本项目采用time()函数获得本机时间值。

//027.c 获得本机时间

#include <stdio.h>

#include "time.h"

#include "string.h"

#define MAXDATASIZE 100

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

{

time\_t currentTime;

char timebuffer[MAXDATASIZE];

memset(timebuffer, 0,MAXDATASIZE);

currentTime = time(NULL);

snprintf(timebuffer, MAXDATASIZE, "%s\n", ctime(&currentTime));

printf("time is %s", timebuffer);

return 0;

}

// ctime功能是 把日期和时间转换为字符串

#### 4.2.2TCP通信客户端daytime\_c.c

该客户端循环向服务端发送字符串信息，并接收服务端返回的信息。

//028.c 向服务端发送信息，并接收返回信息的客户端

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <netdb.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <sys/socket.h>

#define SERVPORT 14001

#define MAXDATASIZE 300

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

{

int sockfd,sendbytes;

char buf[MAXDATASIZE];

struct hostent \*host;

struct sockaddr\_in serv\_addr;

if(argc < 2){

fprintf(stderr,"Please enter the server's hostname!\n");

exit(1);

}

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

perror("gethostbyname");

exit(1);

}

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

perror("socket");

exit(1);

}

serv\_addr.sin\_family=AF\_INET;

serv\_addr.sin\_port=htons(SERVPORT);

serv\_addr.sin\_addr=\*((struct in\_addr \*)host->h\_addr);

bzero(&(serv\_addr.sin\_zero),8);

if(connect(sockfd,(struct sockaddr \*)&serv\_addr,\

sizeof(struct sockaddr))==-1){

perror("connect");

exit(1);

}

if((sendbytes=recv(sockfd,buf,MAXDATASIZE,0))==-1){

perror("recv");

exit(1);

}

buf[sendbytes] = '\0';

printf("received msg from server:%s\n",buf);

while (1)

{

printf("Enter the message : ");

if(fgets(buf, sizeof(buf) - 1, stdin) == NULL)

{

break;

}

if((sendbytes=send(sockfd,buf,strlen(buf),0))==-1){

perror("send");

exit(1);

}

printf("sent %d bytes \n", sendbytes);

}

close(sockfd);

}

#### 4.2.3 TCP通信服务端daytime\_s.c

服务端使用time函数获得本机时间值，客户端连接成功后就向客户端发送时间值。后续操作将客户端发来信息原样回复。

//029.c 接收客户发送信息，并返回信息的服务端

#include <sys/types.h>

#include <sys/socket.h>

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <unistd.h>

#include <netinet/in.h>

#define SERVPORT 14001

#define BACKLOG 10

#define MAX\_CONNECTED\_NO 10

#define MAXDATASIZE 50

int main()

{

struct sockaddr\_in server\_sockaddr,client\_sockaddr;

int sin\_size,recvbytes;

int sockfd,client\_fd;

time\_t currentTime;

char timebuffer[MAXDATASIZE+1];

char buf[MAXDATASIZE];

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

perror("socket");

exit(1);

}

printf("socket success!,sockfd=%d\n",sockfd);

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

server\_sockaddr.sin\_port=htons(SERVPORT);

server\_sockaddr.sin\_addr.s\_addr=INADDR\_ANY;

bzero(&(server\_sockaddr.sin\_zero),8);

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

perror("bind");

exit(1);

}

printf("bind success!\n");

if(listen(sockfd,BACKLOG)==-1){

perror("listen");

exit(1);

}

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

sin\_size = sizeof(struct sockaddr);

if((client\_fd=accept(sockfd,(struct sockaddr \*)&client\_sockaddr,&sin\_size))==-1){

perror("accept");

exit(1);

}

currentTime = time(NULL);

snprintf(timebuffer, MAXDATASIZE, "%s\n", ctime(&currentTime));

if ((recvbytes =write(client\_fd, timebuffer, strlen(timebuffer))) <0 ) {

perror("write");

exit(1);

}

while (1) {

if((recvbytes=recv(client\_fd,buf,MAXDATASIZE,0))==-1){

perror("recv");

exit(1);

}

buf[recvbytes] = '\0';

printf("received msg from clients :%s\n",buf);

}

close(sockfd);

}

请解释 snprintf 函数使用方法。

#### 4.2.4 TCP通信服务端回复时间

请修改服务端程序，服务端每次回复最新时间给客户端。

### 4.3Linux中线程的创建与应用

Linux中多线程编程技术被广泛使用，多线程可以提升程序的运行效率和便利性。多线程编程技术普遍存在应用于较大一点的linux程序中。线程间的通信问题就是线程A怎么把消息传递给线程B。目前线程间通信的用的比较多的主要技术有消息队列、共享内存。在实现线程机制中分为Solaris(较早出现) 和POSIX两种，linux中多线程编程的实现POSIX规范，及利用消息队列进行线程间通信。Linux平台线程函数需要包含头文件:#include<pthread.h> 而Solaris使用的头文件是<thread.h>

线程按照其调度者可以分为用户级线程和核心级线程两种。

（1）用户级线程

用户级线程主要解决的是上下文切换的问题，它的调度算法和调度过程全部由用户自行选择决定，在运行时不需要特定的内核支持。在这里，操作系统往往会提供一个用户空间的线程库，该线程库提供了线程的创建、调度、撤销等功能，而内核仍然仅对进程进行管理。如果一个进程中的某一个线程调用了一个阻塞的系统调用，那么该进程包括该进程中的其他所有线程也同时被阻塞。这种用户级线程的主要缺点是在一个进程中的多个线程的调度中无法发挥多处理器的优势。

（2）核心级线程

这种线程允许不同进程中的线程按照同一相对优先调度方法进行调度，这样就可以发挥多处理器的并发优势。

现在大多数系统都采用用户级线程与核心级线程并存的方法。一个用户级线程可以对应一个或几个核心级线程，也就是“一对一”或“多对一”模型。这样既可满足多处理机系统的需要，也可以最大限度地减少调度开销。

Linux线程在核内是以轻量级进程的形式存在的，拥有独立的进程表项，而所有的创建、同步、删除等操作都在核外pthread库中进行。pthread 库使用一个管理线程（\_\_pthread\_manager()，每个进程独立且唯一）来管理线程的创建和终止，为线程分配线程ID，发送线程相关的信号（比如Cancel），而主线程（pthread\_create()）的调用者则通过管道将请求信息传给管理线程。有关Linux平台中线程编程技术可参考Oracle网站专题讲解。题目是：Multithreaded Progamming Guide 链接地址是：

https://docs.oracle.com/cd/E19455-01/806-5257/index.html

条件变量(condition variablex) 使线程阻塞直到变量状态改变。

互斥量(mutual exclusion locks) 对共享数据的互斥锁。

下面是POSIX的线程库。

pthread\_create();//使用默认的属性创建一个线程

pthread\_arrt\_init();//创建一个属性对象控制线程的属性。

pthread\_arrt\_t tarrt;

pthread\_join();//等待一个线程结束并回收线程资源

pthread\_detach();//当线程结束时，加收其资源

单线程的C程序有两种数据：局部数据和全局数据，多线程C程序增加了一个：线程专属数据(TSD)。

pthread\_key\_create();//创建一个key，用于标识线程专属数据。创建无须同步，但是存取须同步控制。

pthread\_key\_t key;

pthread\_key\_delete();//销毁一个key，程序员应负责在销毁前释放key关联的资源。

pthread\_setspecific();//对key进行绑定。

pthread\_getspecific();//获取key绑定的值。

pthread\_self();//获取线程标识

sched\_yield();//同级或更高级别的线程使当前线程退出。

pthread\_setschedparam();//设置当前线程的优先级。

pthread\_getschedparam();//获取当前线程的优先级。

pthread\_kill();//向线程发送信号 线程必须在同一进程内。

int pthread\_kill(thread\_t tid, int sig);  
#include<pthread.h>  
#include<signal.h>  
int sig;  
pthread\_t tid;  
int ret;  
ret =pthread\_kill(tid, sig);

pthread\_exit();//使当前线程终止

pthread\_cancel();//取消一个线程的执行。

pthread\_setcancelstate();//使一个线程可取消或不可取消。

pthread\_testcancel();//创建一个取消点。

pthread\_arrt\_init();//设置属性变量值，用于控制线程。

Creating a Detached Thread //创建一个分享属生的线程  
#include <pthread.h>  
pthread\_attr\_t tattr;  
pthread\_t tid;  
void \*start\_routine;  
void arg  
int ret;  
  
/\* initialized with default attributes \*/  
ret = pthread\_attr\_init()(&tattr);  
ret = pthread\_attr\_setdetachstate()(&tattr,PTHREAD\_CREATE\_DETACHED);  
ret = pthread\_create()(&tid, &tattr, start\_routine, arg);

pthread\_attr\_setstatcksize();//设置线程堆栈大小。

在Linux平台中线程操作相关函数有:

1.线程创建函数pthread\_create

2.等待另一线程结束函数pthread\_join

3.线程分离函数pthread\_detach

4.线程自我结束函数pthread\_exit

多线程程序在运行时还需要执行必要的互斥，线程间也可使用信号量进行通讯。

#### 4.3.1使用pthread\_create创建线程

Linux中创建线程使用pthead\_create函数，

#inlcude<pthread.h>

int pthread\_create();

将下面的代码进行编译，尝试运行。

/\*《TCP/IP网络编程》page287

\* thread1.c

\* 057.c 线程实例1

\*/

#include<stdio.h>

#include<pthread.h>

#include<unistd.h>

void \* thread\_main(void \*arg);

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

{

pthread\_t t\_id;

int thread\_param=5;

if(pthread\_create(&t\_id,NULL,thread\_main,(void\*)&thread\_param)!=0)

{

puts("pthread\_create() error");

return -1;

}

sleep(10);

puts("end of main");

return 0;

}

void \*thread\_main(void \*arg)

{

int i;

int cnt=\*((int\*)arg);

for(i=0;i<cnt;i++)

{

sleep(1);

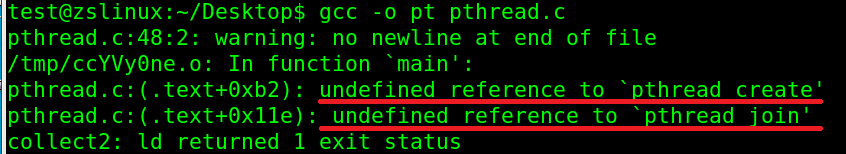
puts("running thread");

}

return NULL;

}

" undefined reference to 'pthread\_create'"，所有关于线程的函数都会有此错误，导致无法编译通过。问题的原因：pthread不是Linux下gcc默认的库，也就是在链接的时候，无法找到phread库中相应函数的入口地址，链接失败。



解决方法：在gcc编译的时候，附加要加 -lpthread参数即可解决。

gcc -o pt pthread.c -lpthread

#### 4.3.2使用pthread\_join等待线程结束

当创建的线程执行完成后资源如何回收呢？默认情况下创建的线程与原有线程属非分离状态，当原有线程调用pthread\_join（）函数来等待创建的线程结束，创建的线程代码执行完毕释放占用的资源。也可以设置被创建的线程为分离的，这样新创建的线程运行结束后系统马上释放其资源。下面看看pthread\_join方法的使用。

/\*《Linux网络编程》page127

\* pthread.c

\* 058.c 线程实例2

\*/

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

static int run = 1; /\*运行状态参数\*/

static int retvalue ; /\*线程返回值\*/

void \*start\_routine(void \*arg) /\*线程处理函数\*/

{

int \*running = arg; /\*获取运行状态指针\*/

printf("子线程初始化完毕，传入参数为:%d\n",\*running); /\*打印信息\*/

while(\*running) /\*当running控制参数有效\*/

{

printf("子线程正在运行\n"); /\*打印运行信息\*/

usleep(1); /\*等待\*/

}

printf("子线程退出\n"); /\*打印退出信息\*/

retvalue = 8; /\*设置退出值\*/

pthread\_exit( (void\*)&retvalue); /\*线程退出并设置退出值\*/

}

int main(void)

{

pthread\_t pt;

int ret = -1;

int times = 3;

int i = 0;

int \*ret\_join = NULL;

ret = pthread\_create(&pt, NULL, (void\*)start\_routine, &run); /\*建立线程\*/

if(ret != 0) /\*建立线程失败\*/

{

printf("建立线程失败\n"); /\*打印信息\*/

return 1; /\*返回\*/

}

usleep(1); /\*等待\*/

for(;i<times;i++) /\*进行3次打印\*/

{

printf("主线程打印\n"); /\*打印信息\*/

usleep(1); /\*等待\*/

}

run = 0; /\*设置线程退出控制值，让线程退出\*/

pthread\_join(pt,(void\*)&ret\_join); /\*等待线程退出\*/

printf("线程返回值为:%d\n",\*ret\_join); /\*打印线程的退出值\*/

return 0;

}

#### 4.3.3线程中使用信号量

编译具有信号量函数的代码也需要使用-lpthread参数

// 059.c 线程间信号量的使用 《TCP/IP网络编程》 page305

#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

void \* read(void \*arg);

void \* accu(void \*arg);

static sem\_t sem\_one;

static sem\_t sem\_two;

static int num;

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

{

pthread\_t id\_t1,id\_t2;

sem\_init(&sem\_one,0,0);

sem\_init(&sem\_two,0,1);

pthread\_create(&id\_t1,NULL,read,NULL);

pthread\_create(&id\_t2,NULL,accu,NULL);

pthread\_join(id\_t1,NULL);

pthread\_join(id\_t2,NULL);

sem\_destroy(&sem\_one);

sem\_destroy(&sem\_two);

return 0;

}

void \*read(void \* arg)

{

int i;

for(i=0;i<5;i++)

{

fputs("Input num: ",stdout);

sem\_wait(&sem\_two);

scanf("%d",&num);

sem\_post(&sem\_one);

}

return NULL;

}

void \*accu(void \* arg)

{

int sum=0,i;

for(i=0;i<5;i++)

{

sem\_wait(&sem\_one);

sum+=num;

sem\_post(&sem\_two);

}

printf("Result: %d \n",sum);

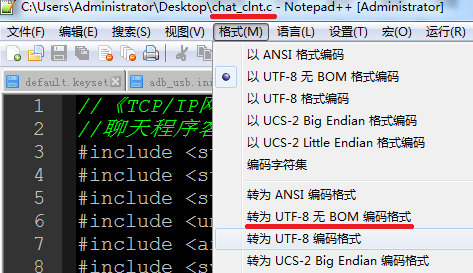
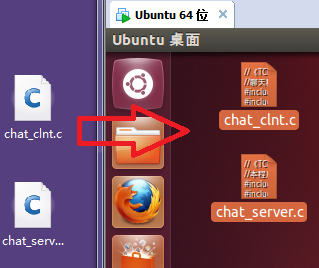
return NULL;

}

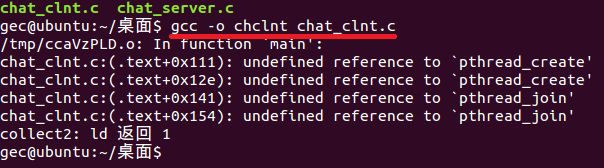
请去掉所有使用信号量的代码，重新运行程序，并对结果进行分析。

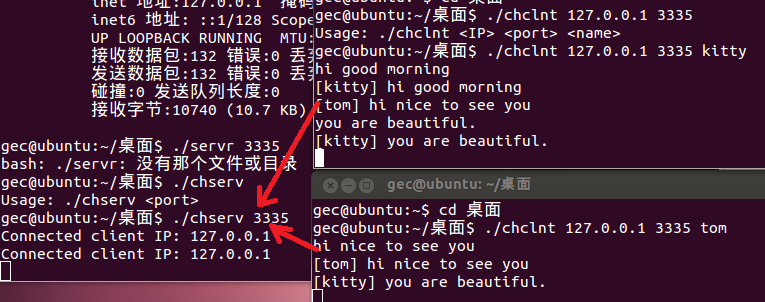
### 4.4使用线程的网络程序

Linux平台还支持通过创建线程执行对每个客户端的socket通讯任务。启动ubuntu18系统，

直接使用gcc命令会出现编译不成功信息。





#### 4.4.1聊天客户端线程分离发送和接收任务

// 060.c 使用线程的聊天客户端《TCP/IP网络编程》page310 chat\_clnt.c

//聊天程序客户端

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <pthread.h>

#define BUF\_SIZE 100

#define NAME\_SIZE 20

void\* recv\_msg(void \*arg);

void\* send\_msg(void \*arg);

void error\_handling(char \*msg);

//全局变量

char name[NAME\_SIZE]="[default]";

char msg[BUF\_SIZE];

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

{

int sock;

struct sockaddr\_in serv\_adr;

pthread\_t snd\_thread,rcv\_thread;

void \* thread\_return;

if(argc!=4)

{

printf("Usage: %s <IP> <port> <name>\n",argv[0]);

exit(1);

}

sprintf(name,"[%s]",argv[3]);

//用于连接和通讯的socket资源

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

memset(&serv\_adr,0,sizeof(serv\_adr));

serv\_adr.sin\_family=AF\_INET;

serv\_adr.sin\_addr.s\_addr=inet\_addr(argv[1]);

serv\_adr.sin\_port=htons(atoi(argv[2]));

//连接目标主机

if(connect(sock,(struct sockaddr\*)&serv\_adr,sizeof(serv\_adr))==-1)

error\_handling("connect() error");

//创建一个线程管理发送数据

pthread\_create(&snd\_thread,NULL,send\_msg,(void \*)&sock);

//创建一个线程管理接收数据

pthread\_create(&rcv\_thread,NULL,recv\_msg,(void \*)&sock);

//等待发送线程结束

pthread\_join(snd\_thread,&thread\_return);

//等待接收线程结束

pthread\_join(rcv\_thread,&thread\_return);

//关闭socket文件资源

close(sock);

return 0;

}

//发送线程代码

void \* send\_msg(void \* arg)

{

int sock=\*((int \*)arg);

char name\_msg[NAME\_SIZE+BUF\_SIZE];

while(1)

{

fgets(msg,BUF\_SIZE,stdin);

if(!strcmp(msg,"q\n")||!strcmp(msg,"Q\n"))

{

close(sock);

exit(0);

}

sprintf(name\_msg,"%s %s",name,msg);

write(sock,name\_msg,strlen(name\_msg));

}

return NULL;

}

//接收信息线程代码

void \*recv\_msg(void \*arg)

{

int sock=\*((int \*)arg);

char name\_msg[NAME\_SIZE+BUF\_SIZE];

int str\_len;

while(1)

{

str\_len=read(sock,name\_msg,NAME\_SIZE+BUF\_SIZE-1);

if(str\_len==-1)

return (void\*)-1;

name\_msg[str\_len]=0;

fputs(name\_msg,stdout);

}

return NULL;

}

//输出出错信息后结束本进程

void error\_handling(char \* message)

{

fputs(message,stderr);

fputc('\n',stderr);

exit(1);

}

#### 4.4.2 多线程并发服务器端实现

下面程序的服务端针对每个客户连接请求，服务端都创建一个新线程进行通讯，更具体点描述是用同一个线程代码生成多个线程体，所以每个通讯的逻辑是完全一样的，但每个线程体使用的资源不一样。

// 061.c 使用线程的并发服务端《TCP/IP网络编程》page307 chat\_server.c

//本程序执行socket监听，信息发送操作

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <pthread.h>

#define BUF\_SIZE 100

#define MAX\_CLNT 256

void \* handle\_clnt(void \*arg);

void send\_msg(char \*msg,int len);

void error\_handling(char \*msg);

//全局变量

int clnt\_cnt=0;

int clnt\_socks[MAX\_CLNT];

pthread\_mutex\_t mutx;

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

{

int serv\_sock,clnt\_sock;

struct sockaddr\_in serv\_adr,clnt\_adr;

int clnt\_adr\_sz;

pthread\_t t\_id;

if(argc!=2)

{

printf("Usage: %s <port>\n",argv[0]);

exit(1);

}

pthread\_mutex\_init(&mutx,NULL);

//用于监听的socket资源

serv\_sock=socket(PF\_INET,SOCK\_STREAM,0);

memset(&serv\_adr,0,sizeof(serv\_adr));

serv\_adr.sin\_family=AF\_INET;

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

serv\_adr.sin\_port=htons(atoi(argv[1]));

//绑定监听端口

if(bind(serv\_sock,(struct sockaddr\*)&serv\_adr,sizeof(serv\_adr))==-1)

error\_handling("bind() error");

//执行监听

if(listen(serv\_sock,5)==-1)

error\_handling("listen() error");

while(1)

{

clnt\_adr\_sz=sizeof(clnt\_adr);

//每个客户连接请求，服务端都创建一个新socket资源用于通讯

clnt\_sock=accept(serv\_sock,(struct sockaddr\*)&clnt\_adr,&clnt\_adr\_sz);

pthread\_mutex\_lock(&mutx);

clnt\_socks[clnt\_cnt++]=clnt\_sock;

pthread\_mutex\_unlock(&mutx);

//创建一个线程管理与客户端的socket通讯

pthread\_create(&t\_id,NULL,handle\_clnt,(void \*)&clnt\_sock);

pthread\_detach(t\_id);

printf("Connected client IP: %s \n",inet\_ntoa(clnt\_adr.sin\_addr));

}

//关闭监听socket文件资源

close(serv\_sock);

return 0;

}

//线程代码

void \* handle\_clnt(void \* arg)

{

int clnt\_sock=\*((int \*)arg);

int str\_len=0,i;

char msg[BUF\_SIZE];

while((str\_len=read(clnt\_sock,msg,sizeof(msg)))!=0)

send\_msg(msg,str\_len);

pthread\_mutex\_lock(&mutx);

for(i=0;i<clnt\_cnt;i++)//remove disconnected client

{

if(clnt\_sock==clnt\_socks[i])

{

while(i++<clnt\_cnt-1)//向前移动数据元素

clnt\_socks[i]=clnt\_socks[i+1];

break;

}

}

clnt\_cnt--;

pthread\_mutex\_unlock(&mutx);

close(clnt\_sock);

return NULL;

}

void send\_msg(char \* msg,int len)

{

int i;

pthread\_mutex\_lock(&mutx);

for(i=0;i<clnt\_cnt;i++)

write(clnt\_socks[i],msg,len);

pthread\_mutex\_unlock(&mutx);

}

//输出出错信息后结束本进程

void error\_handling(char \* message)

{

fputs(message,stderr);

fputc('\n',stderr);

exit(1);

}

该服务程序最大的问题是没有设置退出响应，里面是个死循环。

## 五、程序调试中遇到的问题和解决过程及运行结果

1.将上述每个程序先使用gcc 编译并运行，请将程序结果进行截图，提交作业到学习通平台。

2.分析过程中遇到问题，排查程序出错的原因，并将每个程序结果截图保存，排查程序出错原因，描述解决过程。