电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2016220304031

姓 名 邓巧

（实验）课程名称 嵌入式网络编程

理论教师 兰刚

实验教师 冯月

**电 子 科 技 大 学**

**实 验 报 告**

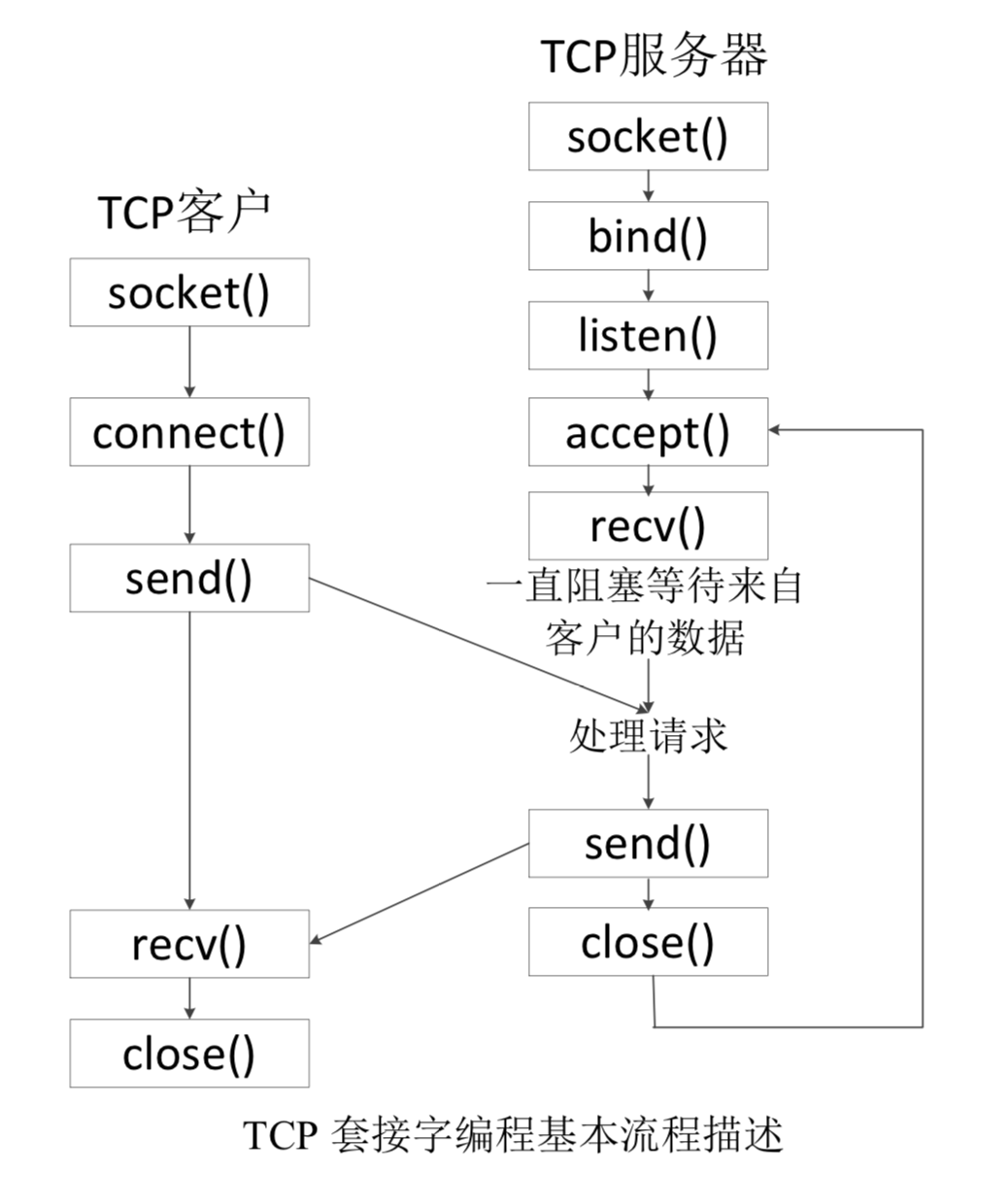
**学生姓名：邓巧 学号：2016220304031 指导教师：冯月**

**实验地点：信软楼304 实验时间：2018.9.25**

1. 实验名称：Linux 平台下的基于 TCP 的文本聊天程序
2. 实验学时：4
3. 实验目的：

掌握 Linux 下的 Socket 编程相关的基本数据结构和 TCP 数据通信的函数使用，实现基于 TCP 数据传输的文本聊天程序。

1. 实验原理：



1. 实验内容：

文本聊天程序的双方分别是服务器方和客户方。

(1)服务器方: 服务器程序等待客户的连接，客户连接成功后可以与客户进行文字通信。服务器如果收到客户方的 exit 文本，则退出服务器程序。

(2)客户方: 建立一个用 TCP 连接的客户，以命令行的方式启动，命令行要求有服务器 地址，服务器端口号。连接建立成功后可以输入文本与服务器进行通信。客户方 通过输入 exit 文本结束语服务器的通信并退出程序。

1. 实验器材（设备、元器件）：

Linux 操作系统、带有 C 语言编译插件的 Eclipse 开发工具。

1. 实验步骤：

1.创建套接字函数

格式：int socket（int domain， int type，int protocol）

功能：根据给定的参数创建一个套接字描述子，成功返回一个套接字描述符，不成功返回-1。

参数：

domain指示网络协议族，AF\_INET或AF\_INET6标识TCP/IP协议；

type指示协议类型，如SOCK\_STREAM（TCP）、SOCK\_DRAM（UDP）和SOCK\_RAW；

protocol指示特定的协议类型，type为SOCK\_STREAM和SOCK\_DGRAM时protocol设置为0，type为SOCK\_RAW时可设置不同的值（IPPROTO\_IP、IPPROTO\_ICMP等）

2.绑定套接字函数

格式：int bind（int sockfd, const struct sockaddr \*addr, socklen\_t addrlen）

功能：绑定IP地址和端口号到套接字sockfd。成功返回0，否则返回-1。

参数：

sockfd指示套接字描述符；

addr指示本地的套接字地址；

addrlen指示套接字地址长度；

3. 建立连接套接字函数

格式：int connect（int sockfd, const struct sockaddr \*addr, socklen\_t addrlen）

功能：建立套接字与给定服务器地址addr的连接。如果sockfd是TCP描述符，则建立一条连接；如果是UDP描述符，则只建立sockfd和addr的绑定关系。成功返回0，否则返回-1。

参数：

sockfd指示套接字描述符；

addr指示服务器的套接字地址；

addrlen指示套接字地址长度；

4. 监听套接字函数

格式：int listen（int sockfd, int backlog）

功能：设置未被接收的TCP连接请求的最大数量，并监听连接请求。成功返回0，否则返回-1。

参数：

sockfd指示套接字描述符；

addr指示套接字地址；

addrlen指示套接字地址长度；

5. 接收请求套接字函数

格式：int accept（int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen）

功能：设置未被接收的TCP连接请求的最大数量，并监听连接请求。成功返回0，否则返回-1。

参数：

sockfd指示套接字描述符；

addr获得对方的套接字地址；

addrlen指示套接字地址长度；

6. 发送数据套接字函数

格式：int send（int sockfd, const void \*buf, size\_t len, int flags）

功能：通过已经连接的套接字发送数据。成功返回发送的字符数，否则返回-1。

参数：

sockfd指示套接字描述符；

buf要发送的数据缓冲区；

len指示要发送数据的长度；

flags指示发送数据的属性，典型的MSG\_OOB指示带外数据发送；

7. 关闭套接字函数

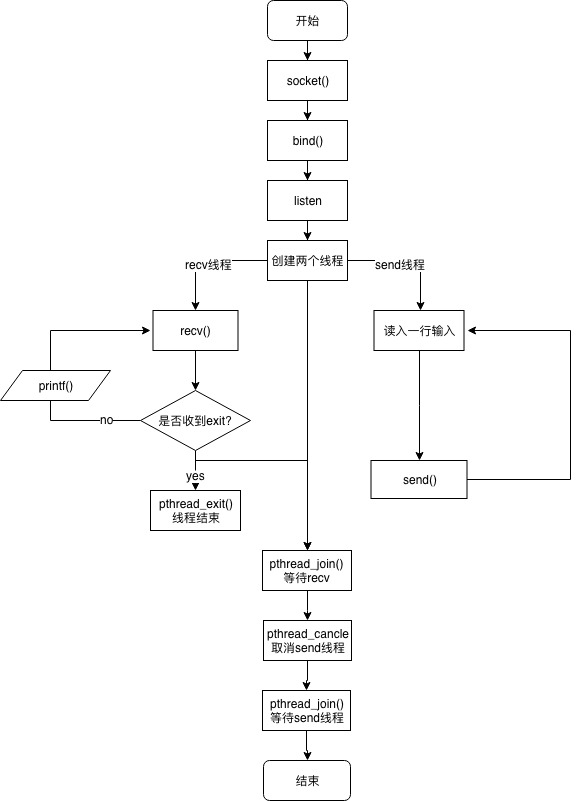
格式：int close（int sockfd）

功能：关闭网络套接字。成功返回0，否则返回-1。

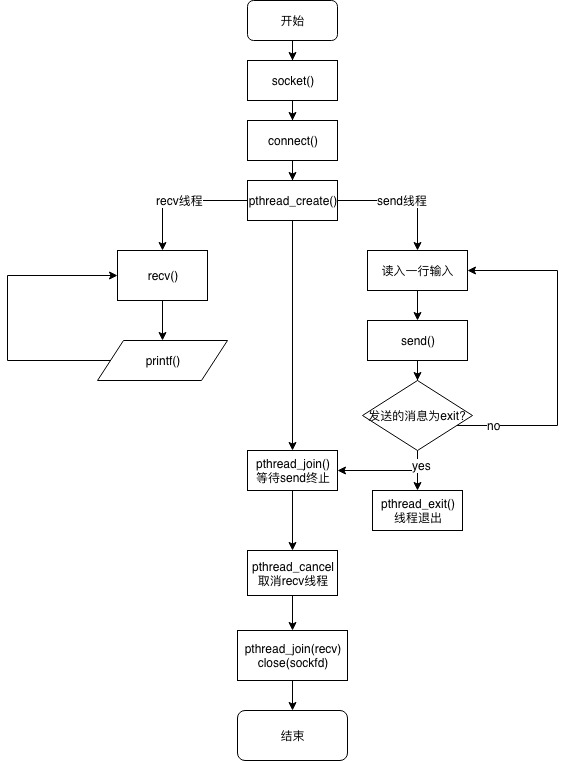
参数：

sockfd指示套接字描述符；

1. 实验结果与分析（含重要数据结果分析或核心代码流程分析）



Server.c程序流程图



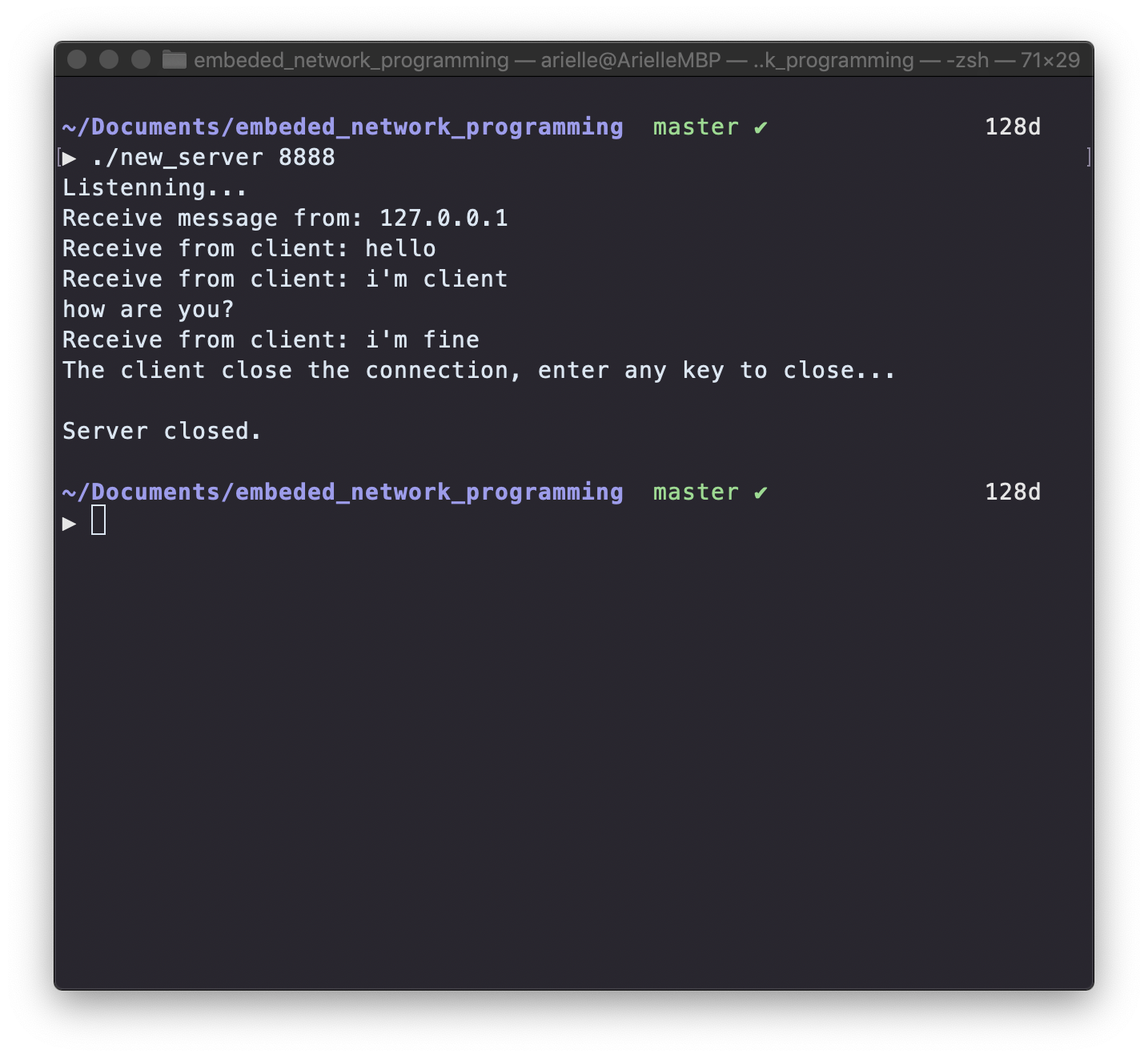
Client.c程序流程图

1. server.c

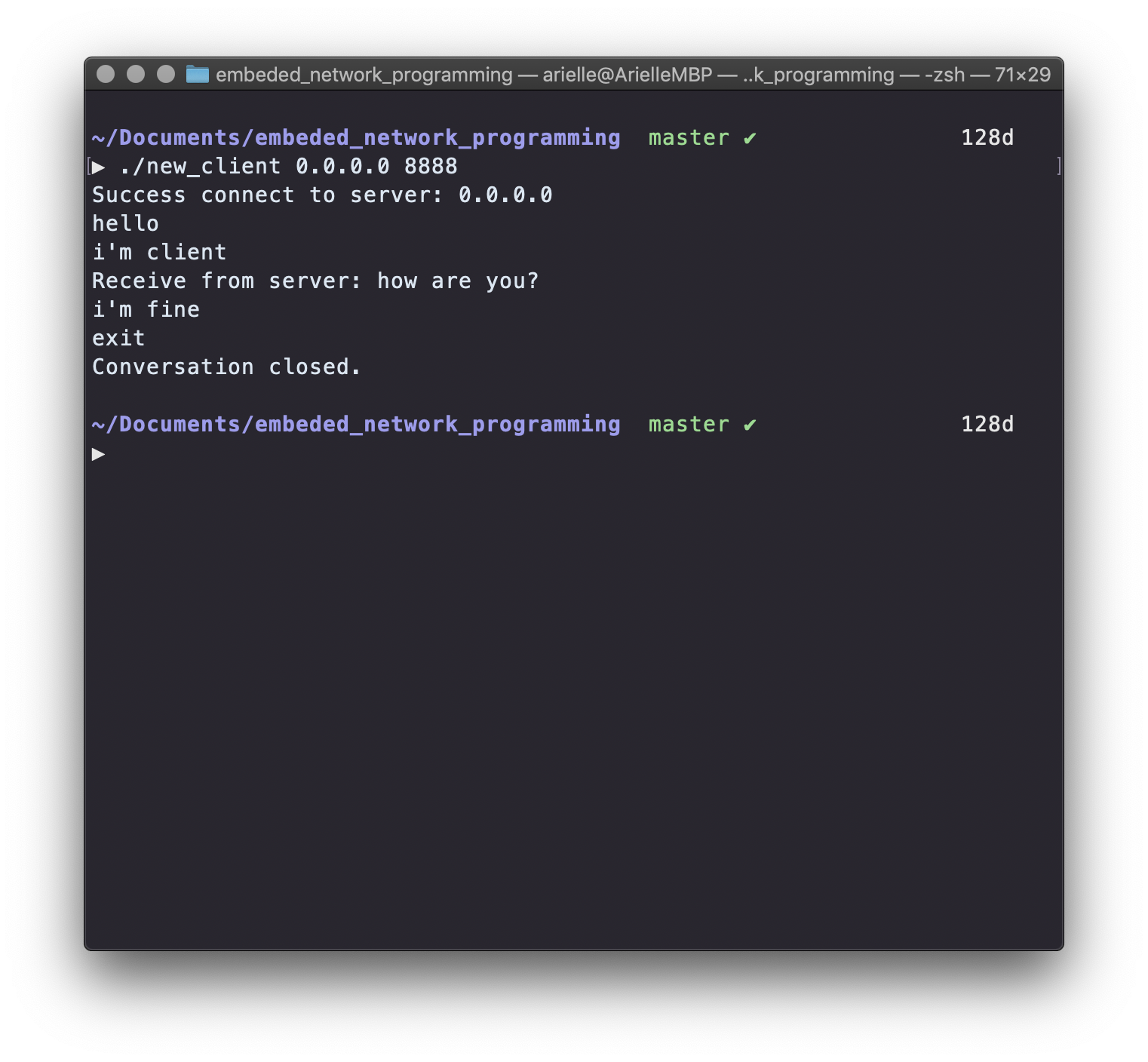
|  |
| --- |
| /\*邓巧 2016220304031\*/  /\*date: 18-9-26\*/  /\*A simple tcp chat program,\*/  /\*Server.c\*/  #include<sys/types.h>  #include<sys/socket.h>  #include<stdio.h>  #include<string.h>  #include<arpa/inet.h>  #include<stdlib.h>  #include <unistd.h>  #include<pthread.h>  #include <signal.h>  #define BUFF\_SIZE 100  int get\_line(char \*buffer,int maxlen);  char exit\_message[] = "exit\n";  void\* recv\_message(void\* socketfd);  void\* send\_message(void\* socketfd);  int main(int argc,char \*argv[]){  void \*retval;  char line[BUFF\_SIZE];  pthread\_t send\_pid,recv\_pid;  int serverfd,clientfd;  char send\_buffer[BUFF\_SIZE];  char recev\_buffer[BUFF\_SIZE];  char ip\_addr[] = "0.0.0.0";  socklen\_t len = sizeof(struct sockaddr);  struct sockaddr\_in server\_addr,client\_addr;  if(!argv[1]){  perror("Lack port number!\n");  exit(EXIT\_FAILURE);  }  if((serverfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1){  perror("Failed to get socketfd!\n");  exit(EXIT\_FAILURE);  }  memset(&server\_addr, 0, sizeof(server\_addr));  server\_addr.sin\_family = AF\_INET;  server\_addr.sin\_port = htons(atoi(argv[1]));  server\_addr.sin\_addr.s\_addr = inet\_addr(ip\_addr);  if(bind(serverfd, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) == -1){  perror("Failed to bind!\n");  exit(EXIT\_FAILURE);  }  if(listen(serverfd,SOMAXCONN) == -1){  perror("Failed to listen!\n");  exit(EXIT\_FAILURE);  }  printf("Listenning...\n");  if((clientfd = accept(serverfd, (struct sockaddr\*) &client\_addr, &len)) == -1){  perror("Failed to accept!\n");  exit(EXIT\_FAILURE);  }  printf("Receive message from: %s\n",inet\_ntoa(client\_addr.sin\_addr));    if(pthread\_create(&send\_pid,NULL,send\_message,(void \*)&clientfd) != 0){  perror("Failed to create send thread!\n");  exit(EXIT\_FAILURE);  }  if(pthread\_create(&recv\_pid,NULL,recv\_message,(void \*)&clientfd) != 0)  perror("Failed to create recv thread!\n");  pthread\_join(recv\_pid,&retval);  if(pthread\_cancel(send\_pid) != 0)  perror("Failed to cancel thread!\n");  pthread\_join(send\_pid,&retval);  get\_line(line,BUFF\_SIZE);  printf("Server closed.\n");  close(serverfd);  exit(EXIT\_SUCCESS);  }  //a safe way to get message consisting blank spaces  int get\_line(char \*line, int max) {  char c;  int len;  for (len = 0; len < max && (c = getchar()) != EOF && c != '\n'; ++len)  line[len] = c;  if (c == '\n') {  line[len] = c;  len++;  }  line[len] = '\0';  return len;  }  void\* recv\_message(void\* socketfd){  void \*retval = NULL;  char recev\_buffer[BUFF\_SIZE];  memset(recev\_buffer,0,BUFF\_SIZE);  while(1){  if(recv(\*((int \*)socketfd),recev\_buffer,sizeof(recev\_buffer),0) > 0){  if(strcmp(recev\_buffer,exit\_message) == 0){  printf("The client close the connection, enter any key to close...\n");  pthread\_exit(retval);  }  printf("Receive from client: %s",recev\_buffer);  memset(recev\_buffer,0,BUFF\_SIZE);  }  }  }  void\* send\_message(void\* socketfd){  char send\_buffer[BUFF\_SIZE];  memset(send\_buffer,0,BUFF\_SIZE);  if(pthread\_setcancelstate(PTHREAD\_CANCEL\_ENABLE, NULL) != 0)  perror("Failed to set cancel!\n");  while(1){  if(get\_line(send\_buffer,BUFF\_SIZE) > 0){  send(\*((int \*)socketfd),send\_buffer,strlen(send\_buffer),0);  memset(send\_buffer,0,BUFF\_SIZE);  }  pthread\_testcancel();  }  } |

1. **clent.c**

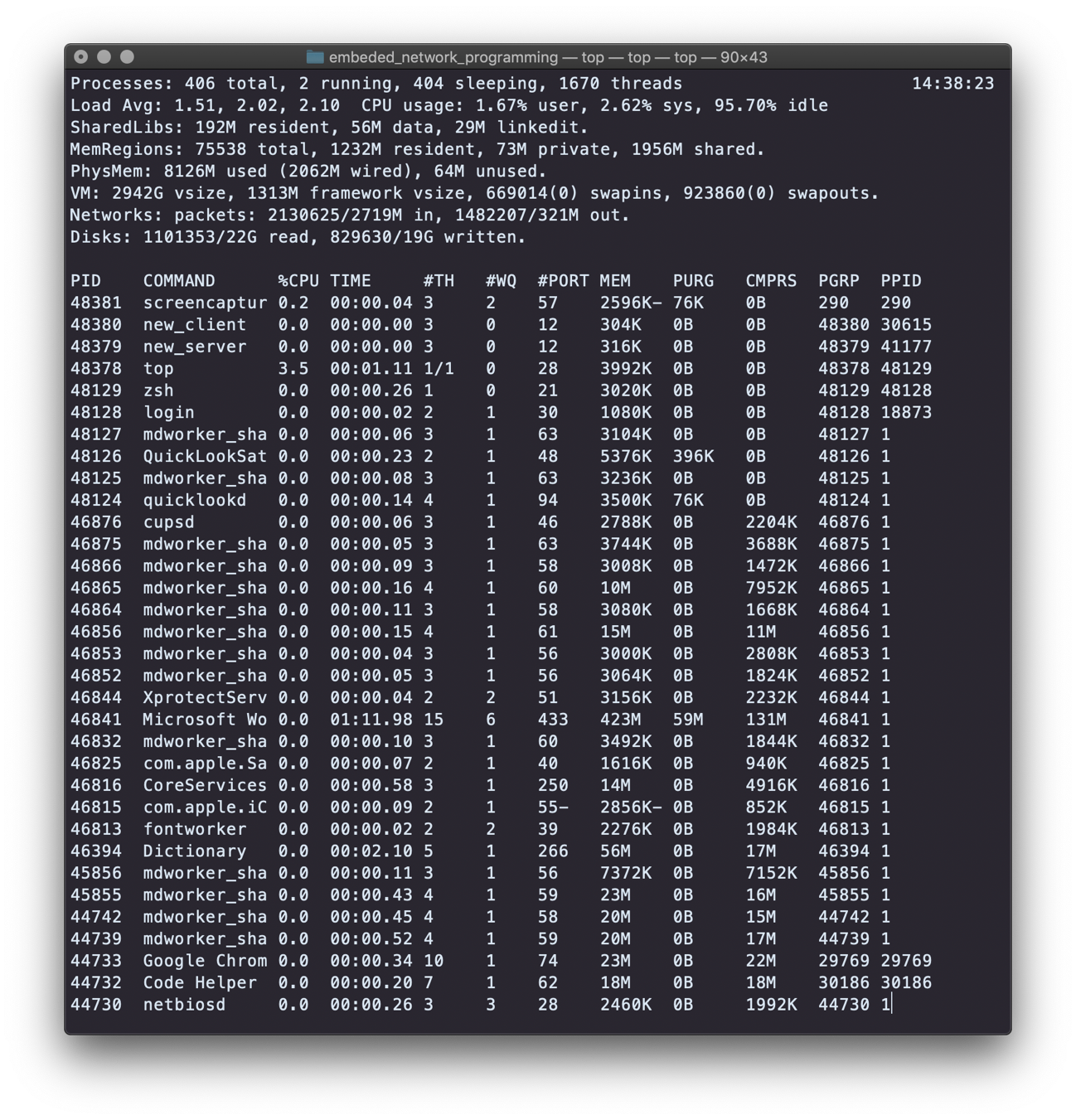
|  |
| --- |
| /\*邓巧 2016220304031\*/  /\*date: 18-9-26\*/  /\*A simple tcp chat program\*/  /\*Client.c\*/  #include<string.h>  #include<stdlib.h>  #include<sys/types.h>  #include<sys/socket.h>  #include<stdio.h>  #include<arpa/inet.h>  #include<unistd.h>  #include<pthread.h>  #include<signal.h>  #define BUFF\_SIZE 100  int get\_line(char \*buffer,int maxlen);  char exit\_message[] = "exit\n";  void\* send\_message(void\* socketfd);  void\* recv\_message(void\* socketfd);  int main(int argc, char \*argv[]){  void\* ret;  pthread\_t send\_pid,recv\_pid;  if(argc != 3){  perror("Lack port and address!\n");  exit(EXIT\_FAILURE);  }  int serverfd,clientfd;  char send\_buffer[BUFF\_SIZE];  char recev\_buffer[BUFF\_SIZE];  struct sockaddr\_in server\_addr;  if((clientfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1){  perror("Failed to get socketfd!\n");  exit(EXIT\_FAILURE);  }  memset(&server\_addr, 0, sizeof(server\_addr));  server\_addr.sin\_family = AF\_INET;  server\_addr.sin\_port = htons(atoi(argv[2]));  server\_addr.sin\_addr.s\_addr = inet\_addr(argv[1]);  if(connect(clientfd,(struct sockaddr\*)&server\_addr,sizeof(struct sockaddr)) == -1){  perror("Failed to connect!\n");  exit(EXIT\_FAILURE);  }  printf("Success connect to server: %s\n",argv[1]);  if(pthread\_create(&send\_pid,NULL,send\_message,(void \*)&clientfd) != 0){  perror("Failed to create send pthread!\n");  exit(EXIT\_FAILURE);  }  if(pthread\_create(&recv\_pid,NULL,recv\_message,(void \*)&clientfd) != 0){  perror("Failed to create recv pthread!\n");  }  pthread\_join(send\_pid,&ret);  if(pthread\_cancel(recv\_pid) != 0){  perror("Failed to cancel thread!\n");  }  pthread\_join(recv\_pid,&ret);  printf("Conversation closed.\n");  close(clientfd);  exit(EXIT\_SUCCESS);  }  //a safe way to get message consisting blank spaces  int get\_line(char \*line, int max) {  char c;  int len;  for (len = 0; len < max && (c = getchar()) != EOF && c != '\n'; ++len)  line[len] = c;  if (c == '\n') {  line[len] = c;  len++;  }  line[len] = '\0';  return len;  }  void\* recv\_message(void\* socketfd){  char recev\_buffer[BUFF\_SIZE];  memset(recev\_buffer,0,BUFF\_SIZE);  if(pthread\_setcancelstate(PTHREAD\_CANCEL\_ENABLE, NULL) != 0){  perror("Failed to set cancel!\n");  }  while(1){  pthread\_testcancel();  if(recv(\*((int \*)socketfd),recev\_buffer,sizeof(recev\_buffer),0) > 0){  printf("Receive from server: %s",recev\_buffer);  memset(recev\_buffer,0,BUFF\_SIZE);  }  }  }  void\* send\_message(void \*socketfd){  void \*retval = NULL;  char send\_buffer[BUFF\_SIZE];  memset(send\_buffer,0,BUFF\_SIZE);  // printf("The pid for recv thread: %d\n",((struct parameter\*)send\_parameter)->pid);  while(1){  if(get\_line(send\_buffer,BUFF\_SIZE) > 0){  send(\*((int\*)socketfd),send\_buffer,strlen(send\_buffer),0);  if(strcmp(send\_buffer,exit\_message) == 0){  pthread\_exit(retval);  }  memset(send\_buffer,0,BUFF\_SIZE);  }  }  } |



Server运行结果



Client运行结果



1. 总结及心得体会：
2. Linux手册上的内容最权威，应该多参考。
3. socket是在应用层和传输层之间的一个抽象层，它把TCP/IP层复杂的操作抽象为几个简单的接口供应用层调用已实现进程在网络中通信。
4. gets函数和fgets函数最大的不同是gets函数的缓冲区虽然由用户提供，但是用户无法指定其一次最多读入多少字节的内容。这一点导致gets变成了一个非常危险的函数。应该尽量避免使用。
5. sockaddr和sockaddr\_in包含的数据都是一样的，但他们在使用上有区别：程序员不应操作sockaddr，sockaddr是给操作系统用的。程序员应使用sockaddr\_in来表示地址，sockaddr\_in区分了地址和端口，使用更方便。
6. 对本实验过程及方法、手段的改进建议：

可以适当提高难度，增加多人聊天功能。

报告评分：

指导教师签字：