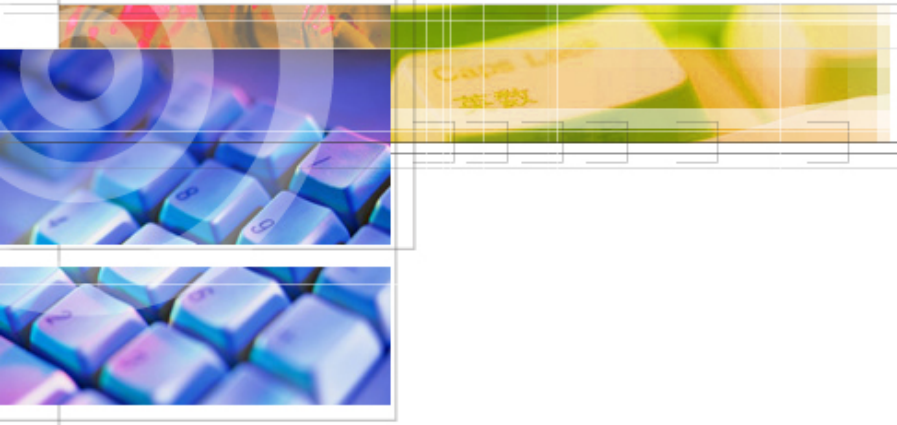


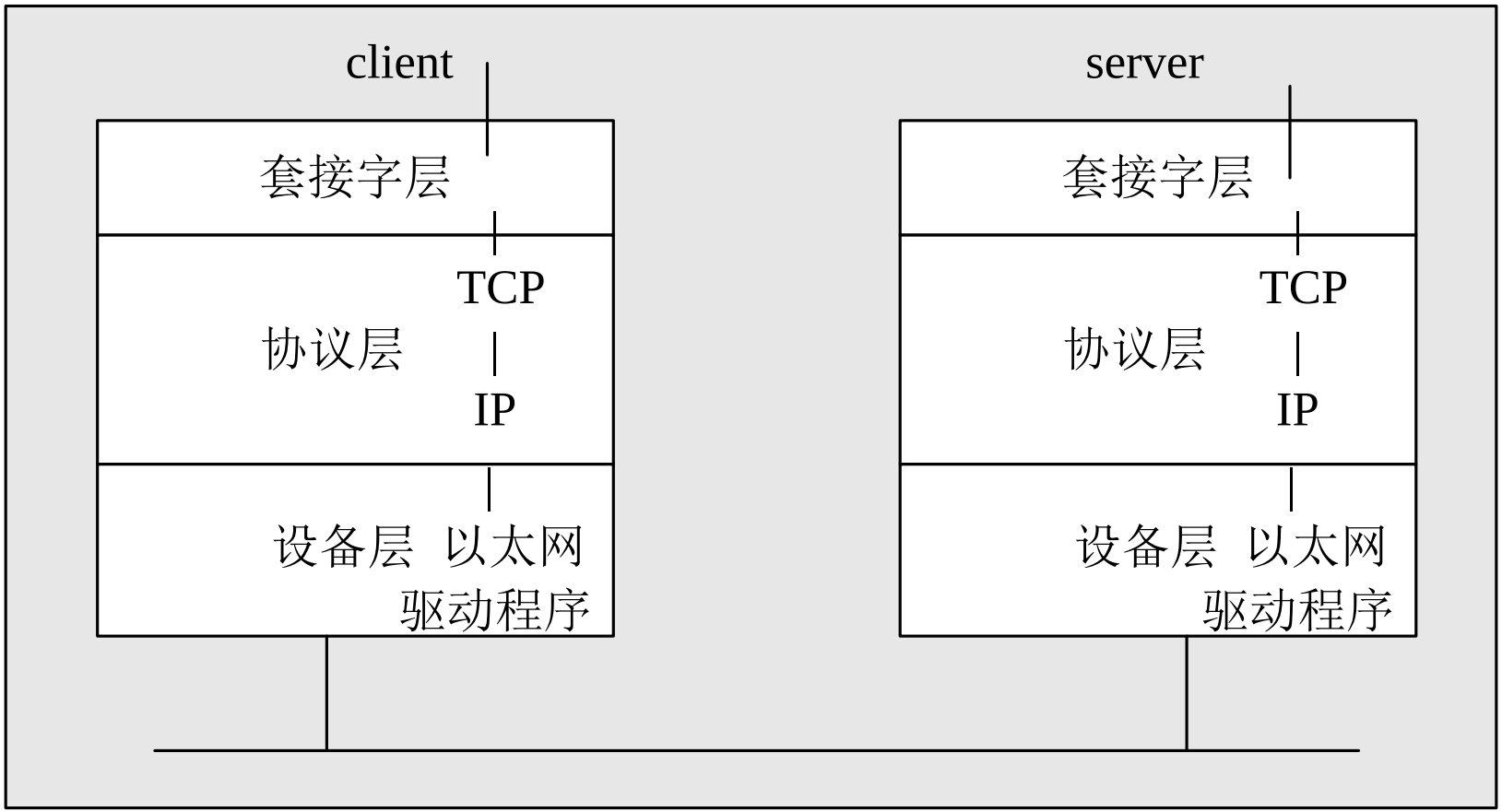
第五讲 嵌入式Linux网络编程

华中科技大学电信学院
鄢舒
E-mail: yan0shu@gmail.com

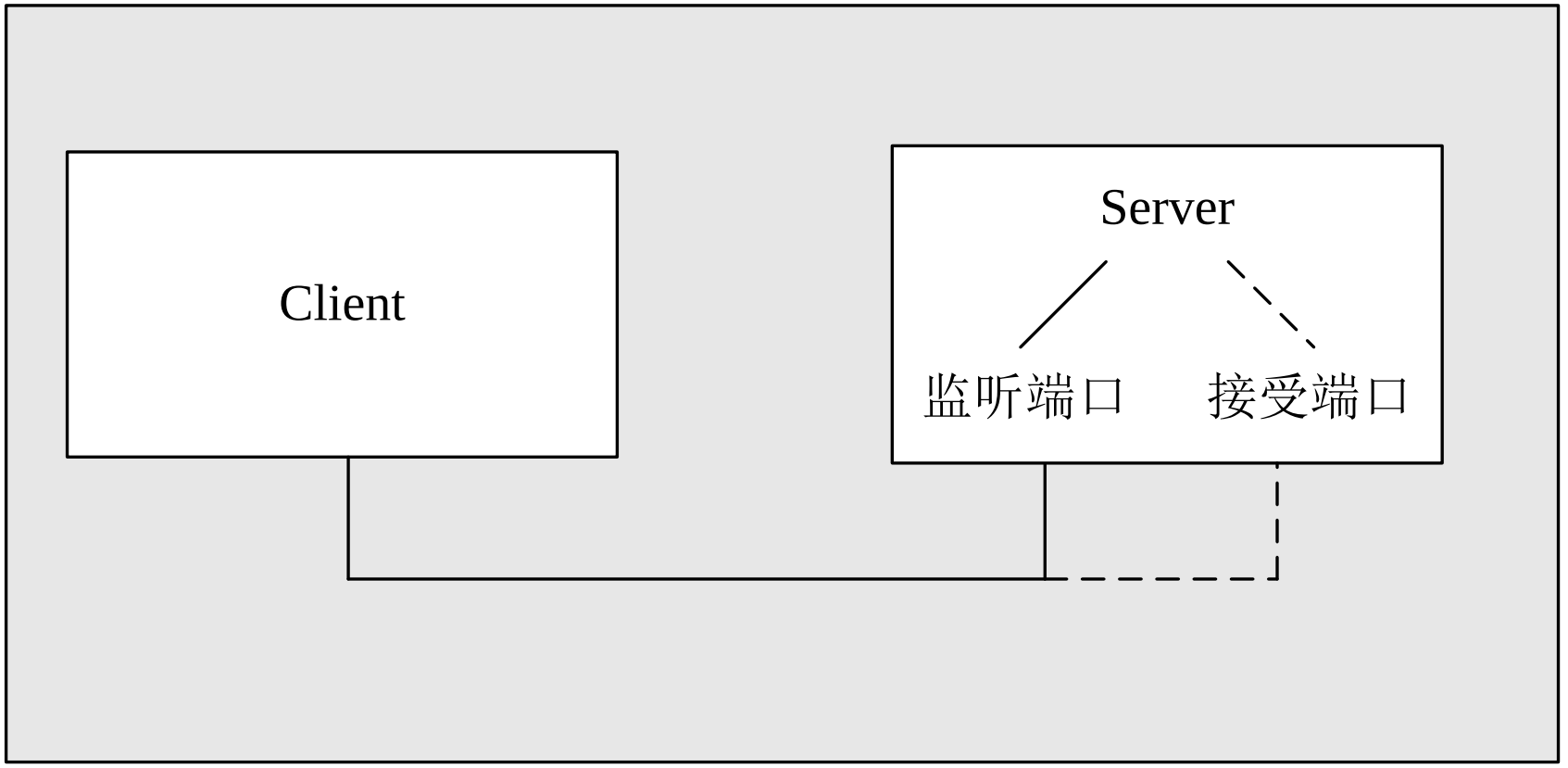
内容提纲(1/5)

- 1. 嵌入式Linux网络编程基础
- 2. socket编程介绍
- 3. socket编程步骤
- 4. socket编程实例
- 5. 实验内容与要求

1.1 Linux网络编程模型



1.2 Linux网络编程端口模型



1.3 TCP与UDP的基本概念

- TCP
  - 面向连接的协议，保证数据传输的正确性，TCP协议发送数据包时，要求对方返回确认数据包，如果没有收到，TCP重新传输数据。
  - 失败数次后，TCP才放弃。
  - 提供流量控制，告诉对方能够接收多少字节数据。
  - 保证数据接收的顺序和发送的顺序一致。
- UDP
  - 无连接协议，不保证数据是否能够到达目的地。
  - 没有流量控制。
  - 不保证数据接收顺序和发送的顺序一致。

内容提纲(2/5)

- 1. 嵌入式Linux网络编程基础
- 2. socket编程介绍
- 3. socket编程步骤
- 4. socket编程实例
- 5. 实验内容与要求

2.1 套接字(socket)概述

- Linux中的网络编程通过socket接口实现。socket既是一种特殊的I/O，也是一种文件描述符。一个完整的socket都有一个相关描述{协议，本地地址，本地端口，远程地址，远程端口}；每一个socket 有一个本地的唯一socket 号，由操作系统分配。

2.2 套接字的类型

套接字有三种类型：

- 流式套接字(SOCK_STREAM)：流式的套接字可以提供可靠的、面向连接的通讯流。它使用了TCP协议。TCP协议保证了数据传输的正确性和顺序性。
- 数据报套接字(SOCK_DGRAM)：数据报套接字定义了一种无连接的服务，数据通过相互独立的报文进行传输，是无序的，并且不保证可靠和无差错。使用数据报协议(UDP协议)。
- 原始套接字：原始套接字允许对低层协议如IP或ICMP直接访问，主要用于新的网络协议实现的测试等。

2.3 套接字地址结构(1/2)

```
struct sockaddr {
  unsigned short sa_family; /* address族, AF_xxx */
  char sa_data[14];      /* 14 bytes的协议地址 */
};
```

- sa_family：一般来说， IPV4使用“AF_INET”。
- sa_data：包含了一些远程电脑的地址、端口和套接字的数目，它里面的数据是杂溶在一起的。

2.3 套接字地址结构(2/2)

```
struct sockaddr_in {
  short int sin_family; /* Internet地址族 */
  unsigned short int sin_port; /* 端口号 */
  struct in_addr sin_addr; /* Internet地址 */
  unsigned char sin_zero[8]; /* 添0（和struct sockaddr
    一样大小）*/
};
```

- 这两个数据类型是等效的，可以相互转换，通常使用sockaddr_in更为方便

2.4 字节序列转换

- 因为各种系统内部对变量的字节存储顺序不同（有的系统是高位在前，低位在后，而有的系统是低位在前，高位在后)，而网络传输的数据是一定要统一顺序的。所以对于内部字节表示顺序和网络字节顺序不同的系统，就一定要对数据进行转换。

2.5 字节转换函数

socket字节转换函数如下：

- htons()——“Host to Network Short” 主机字节顺序转换为网络字节顺序（对无符号短型进行操作2bytes）
- htonl()——“Host to Network Long”主机字节顺序转换为网络字节顺序（对无符号长型进行操作4bytes）
- ntohs()——“Network to Host Short”网络字节顺序转换为主机字节顺序（对无符号短型进行操作2bytes）
- ntohl()——“Network to Host Long ”网络字节顺序转换为主机字节顺序（对无符号长型进行操作4bytes）

2.6 地址格式转换

- Linux提供将点分格式的地址转于长整型数之间的转换函数。如： inet_addr()能够把一个用数字和点表示IP 地址的字符串转换成一个无符号长整型。
- inet_ntoa()（“ntoa”代表“Network to ASCII”）；
- 其他类似函数包括：inet_aton, inet_ntoa, inet_addr等。

2.7 基本套接字调用

socket()　　bind()　　connect()

listen()　　accept()　　send()

recv()　　sendto()　　shutdown()

recvfrom()　　close()　　getsockopt()

setsockopt()　　getpeername()

getsockname()　　gethostbyname()

gethostbyaddr()　　getprotobyname()

fcntl()

内容提纲(3/5)

- 1. 嵌入式Linux网络编程基础
- 2. socket编程介绍
- 3. socket编程步骤
- 4. socket编程实例
- 5. 实验内容与要求

3.1 服务器的套接口编程步骤

- 一个简单服务器的套接口编程步骤
  - 建立套接口
  - 绑定地址和端口
  - 建立套接口队列
  - 接收连接
  - 处理连接
  - 关闭套接口

3.2 客户程序套接口编程的步骤

- 客户程序套接口编程的步骤
  - 建立套接口
  - 连接服务器
  - 处理连接
  - 关闭套接口

3.3 套接口的建立

int socket (int domain, int type, int protocol);

参数说明：

- domain：协议族，可以是下列类型之一
  AF_INET
  AF_UNIX
  AF_LOCAL
- type：套接口的类型
  SOCKET_STREAM 提供TCP套接字
  SOCKET_DGRAM 提供UDP套接字
  SOCKET_RAW 提供原始套接字，允许访问IP层数据包
- protocol：指示对套接字应使用哪个协议。除了使用原始套接口外，protocol通常设置为0
- 返回值：如果调用成功，返回套接字句柄

3.4 bind函数

bind函数：为了使其他的进程能够使用套接口，必须与一个端口联系起来。

int bind (int socket, const struct sockaddr *
address, socklen_t address_len);

参数说明

- socket：调用函数socket()的返回值。
- address：存储IP地址和端口号的数据结构。
- address_len：第二个结构参数的长度。
- 返回值：正确调用返回0，否则返回-1。

3.5 建立套接口队列

int listen (int socket, int backlog);

参数说明：

- socket：调用socket()函数的返回值。
- backlog：指定socket队列的最大连接数，当队列中待处理的连接数超过该值，则拒绝连接。对于一个比较繁忙的网络服务，需要指定较大的backlog值。
- 返回值：正确调用返回0，否则返回-1。

3.6 等待连接函数

int accept (int socket, struct sockaddr *address,
socklen_t *address_len);

参数说明：

- socket：调用socket()函数的返回值。
- address：存放客户程序IP地址和端口号的结构，如果对其不感兴趣，可以设为NULL。
- address_len：第二个结构参数的长度。
- 返回值：返回一个新连接的套接口描述句柄。

3.7 连接服务器函数

int connect (int socket, const struct *address,
socklen_t address_len);

参数说明：

- socket：调用socket()函数的返回值。
- address：存储服务器IP地址和端口的结构。
- address_len：第二个结构参数的长度。
- 返回值：成功调用返回0，否则返回-1。

3.8 TCP接收/发送函数

int recv (int socket, void *buf, size_t len, int flag);

int send (int socket, const void *buf, size_t len, int flags);

参数说明：

- socket：调用socket()函数的返回值。
- buf：缓存区首地址。
- len：期望接收/发送的字节数。
- flags：通信参数，具体值如下：

  MSG_OOB：指示发送/接收的数据为紧急数据

  MSG_PEEK：接收数据，但不把数据从接收缓冲区中删除，调用该函数后，数据仍然可以被以后的读操作获得。
- 返回值：成功调用返回实际接收/发送字节数，否则返回-1。

3.9 UDP接收/发送函数

int recvfrom(int socket, void *buf, size_t len, int flag, struct sockaddr *from, int *fromlen);

int sendto(int socket, const void *buf, size_t len, int flags, const struct sockaddr *to, int tolen);

参数说明：

- from/to 指向一个套接字地址，即通信对端地址
- fromlen/tolen 上面地址参数的长度
- 返回值：成功调用返回实际接收/发送字节数，否则返回-1。

3.10 其他数据通信函数

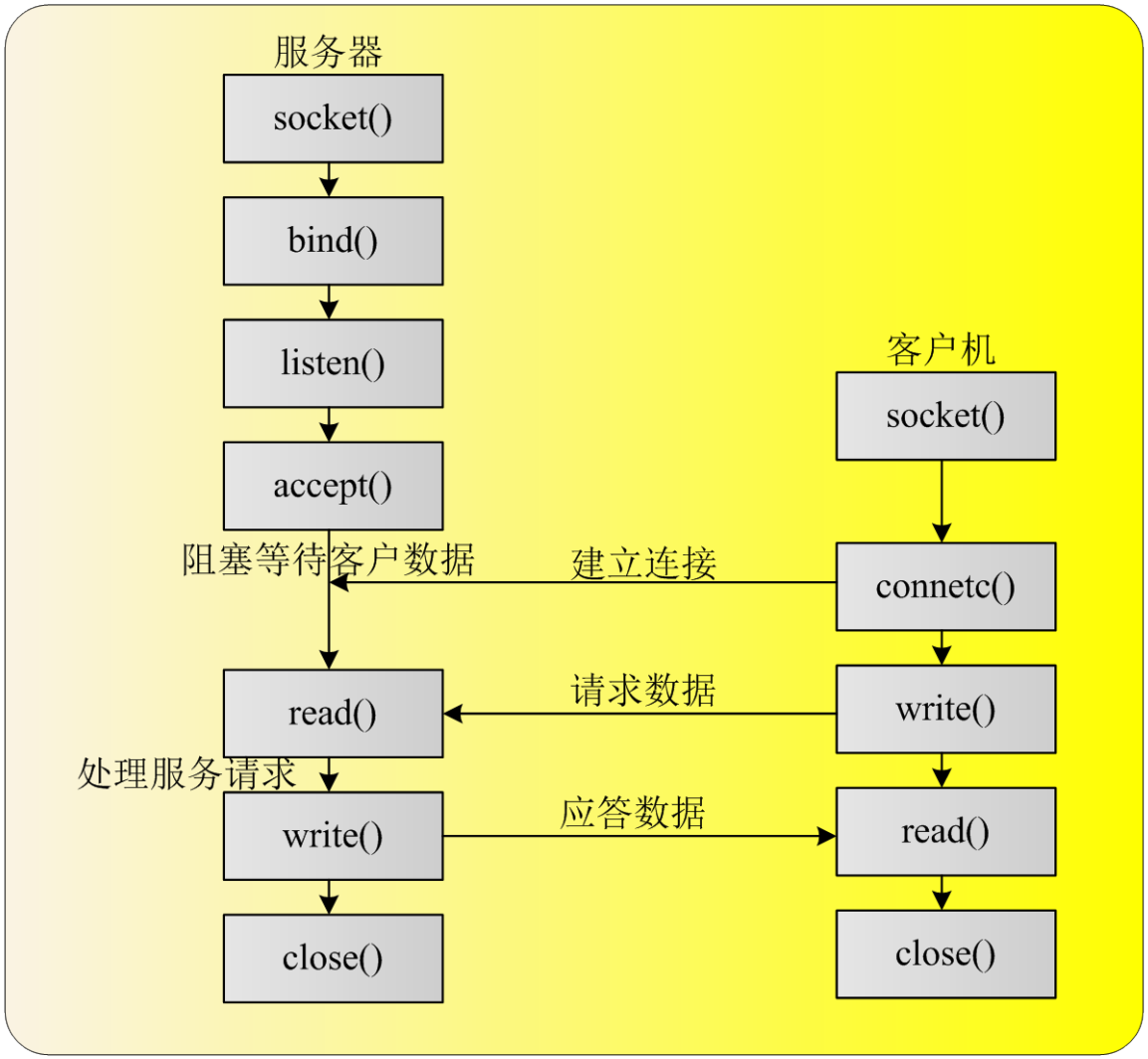
- read：从套接口中读取数据。
- write：从套接口中写入数据。
- close：关闭套接口。

内容提纲(4/5)

- 1. 嵌入式Linux网络编程基础
- 2. socket编程介绍
- 3. socket编程步骤
- 4. socket编程实例
- 5. 实验内容与要求

4.1 TCP/IP套接字编程实例

4.2 TCP编程流图


服务器
socket()
bind()
listen()
accept()
阻塞等待客户数据
read()
处理服务请求
write()
close()
客户机
socket()
建立连接
connetc()
请求数据
write()
应答数据
read()
close()

4.3 SocketHeader.h

```
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>

void print_error(char* error_info)
{
  printf("%s call failed\n", error_info);
  exit(1);
}
```

4.4 TCPServer.c

```
#include "SocketHeader.h"
int main(void)
{
 struct sockaddr_in local; int s; int s1;  int rc; char buf[1];
 local.sin_family = AF_INET; local.sin_port = htons(12345);
 local.sin_addr.s_addr =htonl(INADDR_ANY); s = socket(AF_INET, SOCK_STREAM, 0);
 if(s < 0) print_error("sock");
 rc = bind(s, (struct sockaddr*)&local, sizeof(local));
 if(rc < 0) print_error("bind");
 rc = listen(s,5);
 if(rc < 0)  print_error("listen");
 s1 = accept(s, NULL, NULL);
 if(s1 < 0) print_error("accept");
 rc = recv(s1, buf, 1,0);
 if(rc <= 0) print_error("recv");
 printf("%c\n", buf[0]);
 rc = send(s1, "2", 1, 0);
 if(rc <= 0) print_error("send");
 return 0;
}
```

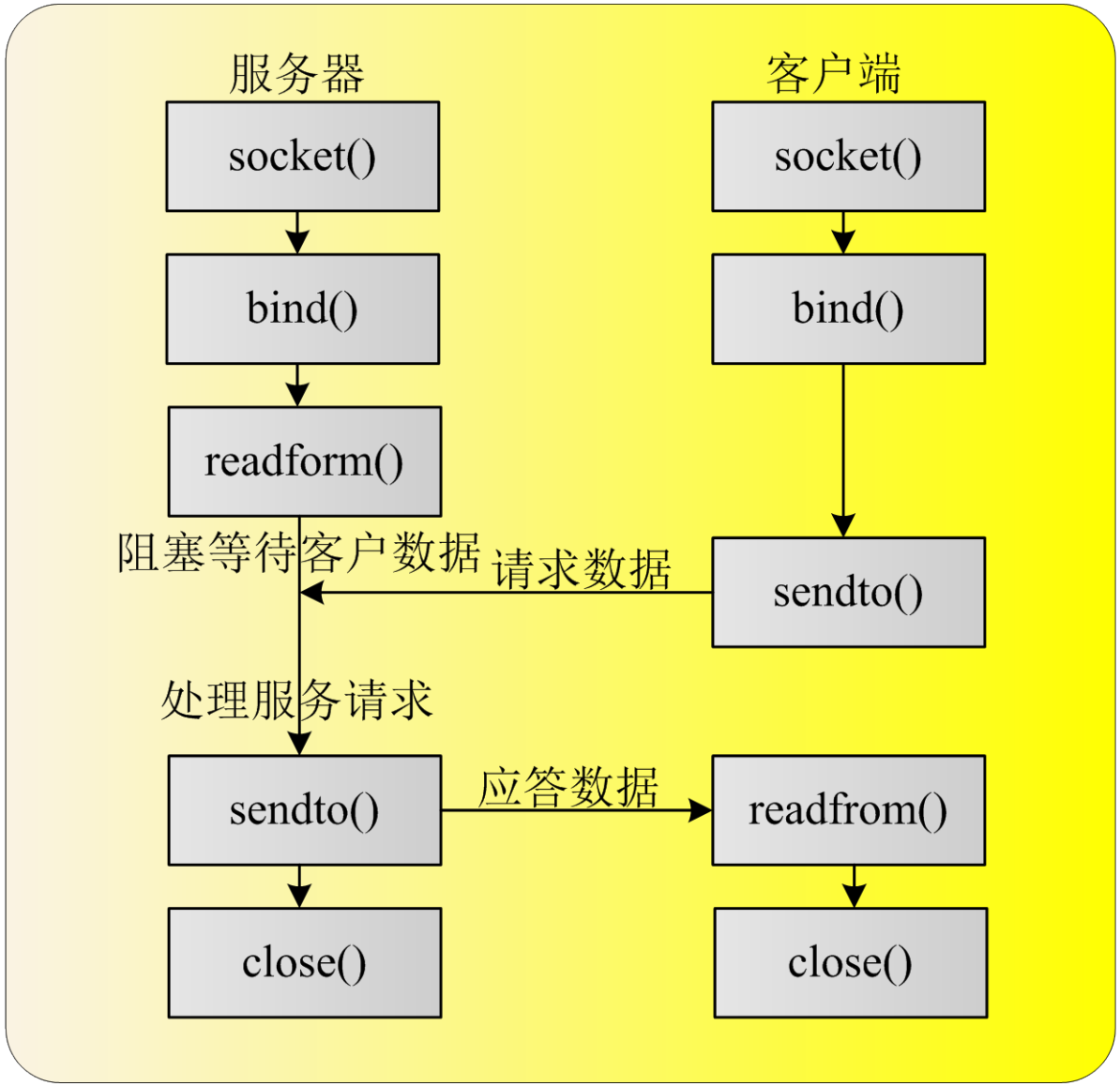
4.5 TCPClient.c

```
#include "SocketHeader.h"
int main(void)
{
 struct sockaddr_in peer; int s; int rc; char buf[1];
 peer.sin_family = AF_INET;  peer.sin_port = htons(12345);
 peer.sin_addr.s_addr = inet_addr("127.0.0.1");
 s = socket(AF_INET, SOCK_STREAM, 0);
 if(s<0) print_error("socket");
 rc = connect(s, (struct sockaddr*)&peer,sizeof(peer));
 if(rc) print_error("connect");
 rc = send(s, "1", 1, 0);
 if(rc <= 0) print_error("send");
 rc = recv(s, buf, 1, 0);
 if(rc <= 0) print_error("recv");
 else printf("%c\n", buf[0]);
 return 0;
}
```

4.6 编译运行TCP例子

- gcc -Wall -o TCPServer TCPServer.c
- gcc -Wall -o TCPClient TCPClient.c
- ./TCPServer
- ./TCPClient

4.7 UDP编程流图


服务器
客户端
socket()
socket()
bind()
bind()
readform()
阻塞等待客户数据
请求数据
sendto()
处理服务请求
sendto()
应答数据
readfrom()
close()
close()

4.8 UDPServer.c

```
#include "SocketHeader.h"
int main(void)
{
  struct sockaddr_in local, client; int client_size; int s; int rc; char buf[1];
  s = socket(AF_INET, SOCK_DGRAM, 0);
  if(s <0) print_error("socket");
  local.sin_family = AF_INET; local.sin_addr.s_addr = htonl(INADDR_ANY);
  local.sin_port = htons(12345);
  rc = bind(s, (struct sockaddr*)&local, sizeof(local));
  if(rc < 0) print_error("bind");
  client_size = sizeof(client);
  rc = recvfrom(s, buf, 1, 0, (struct sockaddr*)&client, &client_size);
  if(rc < 0) print_error("recvfrom");
  printf("%c\n", buf[0]);
  rc = sendto(s, "2", 1, 0, (struct sockaddr*)&client, client_size);
  if(rc < 0) print_error("sendto");
  return 0;
}
```

4.9 UDPClient.c

```
#include "SocketHeader.h"
int main(void)
{
  struct sockaddr_in server, local; int s; int rc; int server_size; char buf[1];
  s = socket(AF_INET, SOCK_DGRAM, 0);
  if(s < 0) print_error("socket");
  local.sin_family = AF_INET; local.sin_addr.s_addr = htonl(INADDR_ANY);
  local.sin_port = htons(0);
  rc = bind(s, (struct sockaddr*)&local, sizeof(local));
  if(rc < 0) print_error("bind");
  server.sin_family = AF_INET; server.sin_addr.s_addr = inet_addr("127.0.0.1");
  server.sin_port = htons(12345); server_size = sizeof(server);
  rc = sendto(s, "1", 1, 0, (struct sockaddr*)&server, server_size);
  if(rc <= 0) print_error("sendto");
  rc = recvfrom(s, buf, 1, 0, (struct sockaddr*)&server, &server_size);
  if(rc <= 0) print_error("recvfrom");
  printf("%c\n",buf[0]);
  return 0;
}
```

4.10 编译运行UDP例子

- gcc -Wall -o UDPServer UDPServer.c
- gcc -Wall -o UDPClient UDPClient.c
- ./UDPServer
- ./UDPClient

4.11 网络高级编程实例

- 由于在前面介绍的函数如connet、recv、send都是阻塞性函数，若资源没有准备好，则调用该函数的进程将进入休眠状态，这样无法实现I/O多路复用了，下面介绍两种I/O多路复用的解决方案。

4.12 利用fcntl函数实现(非阻塞方式)

```
#include <sys/types.h>#include <sys/socket.h>#include <sys/wait.h>#include <stdio.h>#include <stdlib.h>
#include <errno.h> #include <string.h> #include <sys/un.h> #include <sys/time.h> #include <sys/ioctl.h>
#include <unistd.h> #include <netinet/in.h> #include <fcntl.h> #include <unistd.h>
#define SERVPORT 3333#define BACKLOG 10
#define MAX_CONNECTED_NO 10#define MAXDATASIZE 100
int main()
{struct sockaddr_in server_sockaddr,client_sockaddr; int sin_size,recvbytes,flags; int sockfd,client_fd;
 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");
 if((flags=fcntl( sockfd, F_SETFL, 0))<0) perror("fcntl F_SETFL"); flags |= O_NONBLOCK;
 if(fcntl( sockfd, F_SETFL,flags)<0) perror("fcntl");
 while(1){sin_size=sizeof(struct sockaddr_in);
   if((client_fd=accept(sockfd,(struct sockaddr*)&client_sockaddr,&sin_size))==-1){
       perror("accept"); exit(1); }
   if((recvbytes=recv(client_fd,buf,MAXDATASIZE,0))==-1){ perror("recv"); exit(1); }
   if(read(client_fd,buf,MAXDATASIZE)<0){ perror("read"); exit(1); }
   printf("received a connection :%s",buf); close(client_fd); exit(1);}
}
```

4.13 利用select函数实现

```
#include <sys/types.h>#include <sys/socket.h>#include <sys/wait.h>#include <stdio.h>#include <stdlib.h>
#include <errno.h>#include <string.h>#include <sys/un.h>#include <sys/time.h>#include <sys/ioctl.h>
#include <unistd.h>#include <netinet/in.h> #define SERVPORT 3333#define BACKLOG 10
#define MAX_CONNECTED_NO 10 #define MAXDATASIZE 100
int main()
{struct sockaddr_in server_sockaddr,client_sockaddr; int sin_size,recvbytes; fd_set readfd;
  fd_set writefd; int sockfd,client_fd; 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"); FD_ZERO(&readfd); FD_SET(sockfd,&readfd);
  while(1){sin_size=sizeof(struct sockaddr_in);
              if(select(MAX_CONNECTED_NO,&readfd,NULL,NULL,(struct timeval *)0)>0){
                if(FD_ISSET(sockfd,&readfd)>0){
                  if((client_fd=accept(sockfd,(struct sockaddr *)&client_sockaddr,&sin_size))==-1){
                              perror("accept");exit(1);}
                  if((recvbytes=recv(client_fd,buf,MAXDATASIZE,0))==-1){perror("recv");exit(1);}
                  if(read(client_fd,buf,MAXDATASIZE)<0){perror("read");exit(1);}
                  printf("received a connection :%s",buf);}/*if*/
                close(client_fd);}/*select*/}/*while*/
}
```

内容提纲(5/5)

- 1. 嵌入式Linux网络编程基础
- 2. socket编程介绍
- 3. socket编程步骤
- 4. socket编程实例
- 5. 实验内容与要求

5.1 本次实验要求

- 学习嵌入式Linux网络编程的基本方法；
- 在实验平台上实现文件的网络传输：
  - 在实验平台和PC机上分别运行服务器和客户端程序，实现把PC机上的文件传输到实验平台上。
- 有能力可选在实验平台和PC机之间互相传输文件；
- 利用NFS方法调试、运行。

5.2 实验注意事项

- 实验平台上的程序必须用交叉编译工具，而PC机上的程序用本地gcc编译；
- 文本文件可以按ASCII传输，但一般文件需要按二进制传输才能保证完全正确。