Sockets

张海宁

贵州大学

hnzhang1@gzu.edu.cn

May 22, 2018

Overview

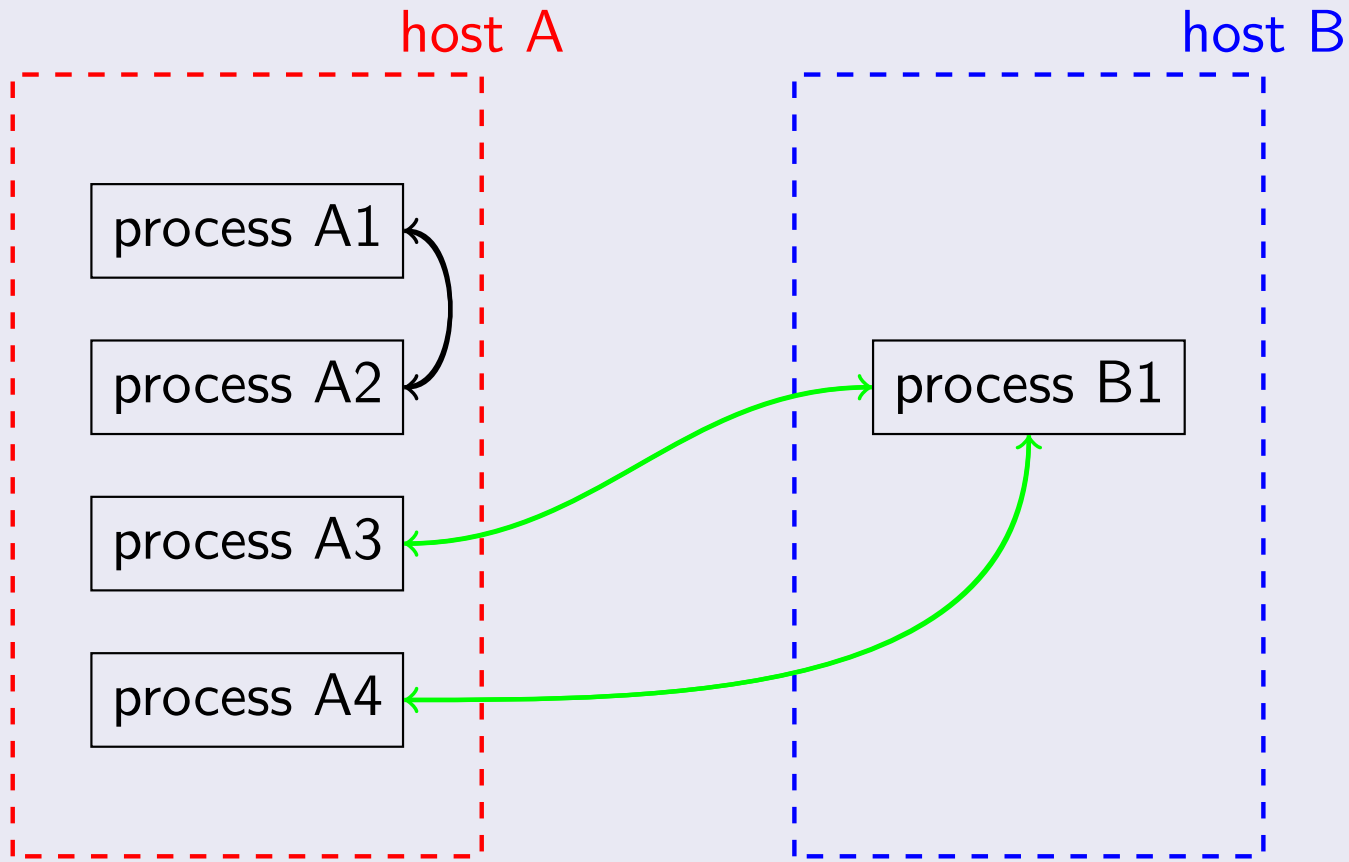
Sockets

What is a Socket

Socket

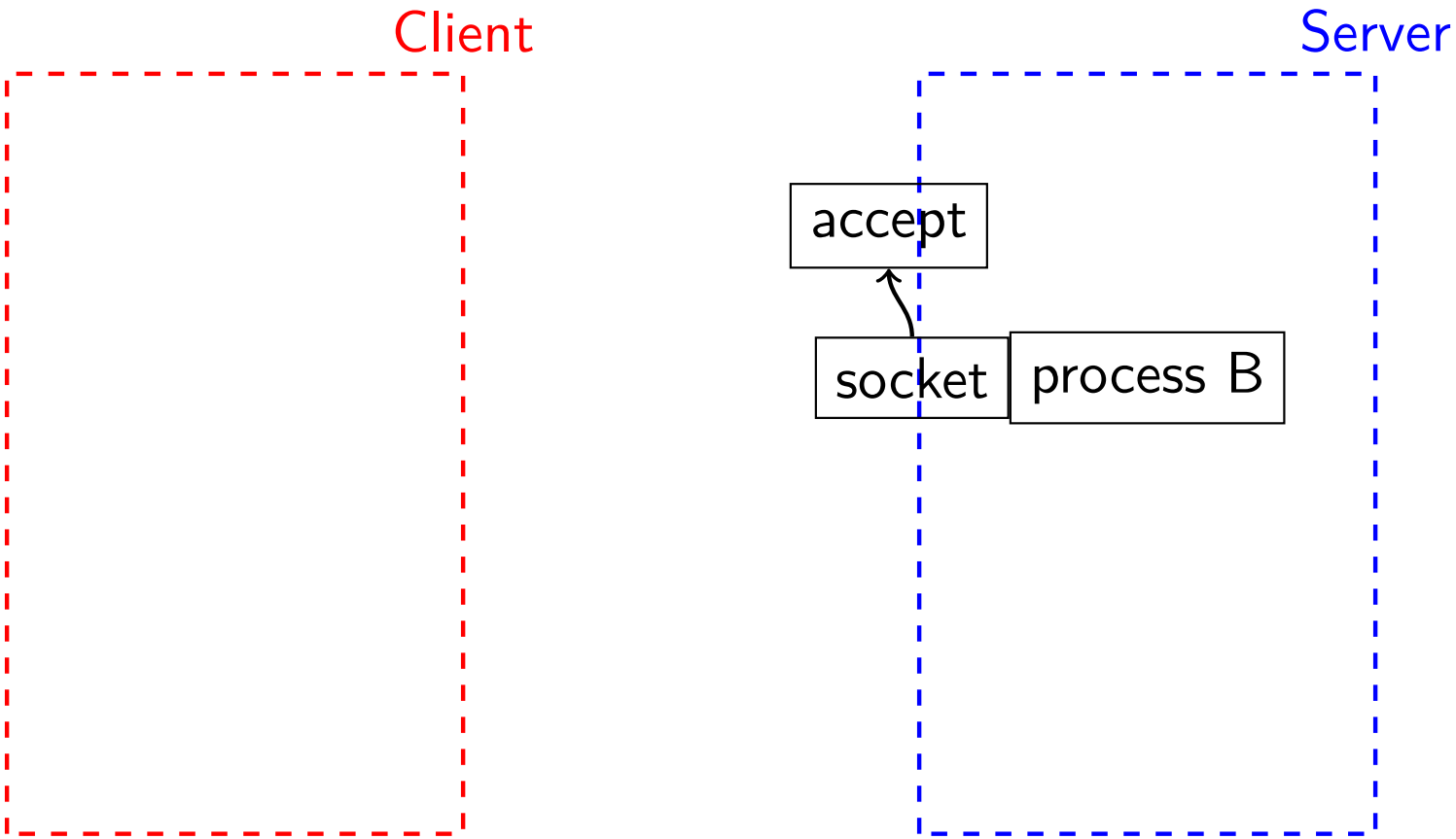
Socket 是一种进程间的通信机制。

进程间通信



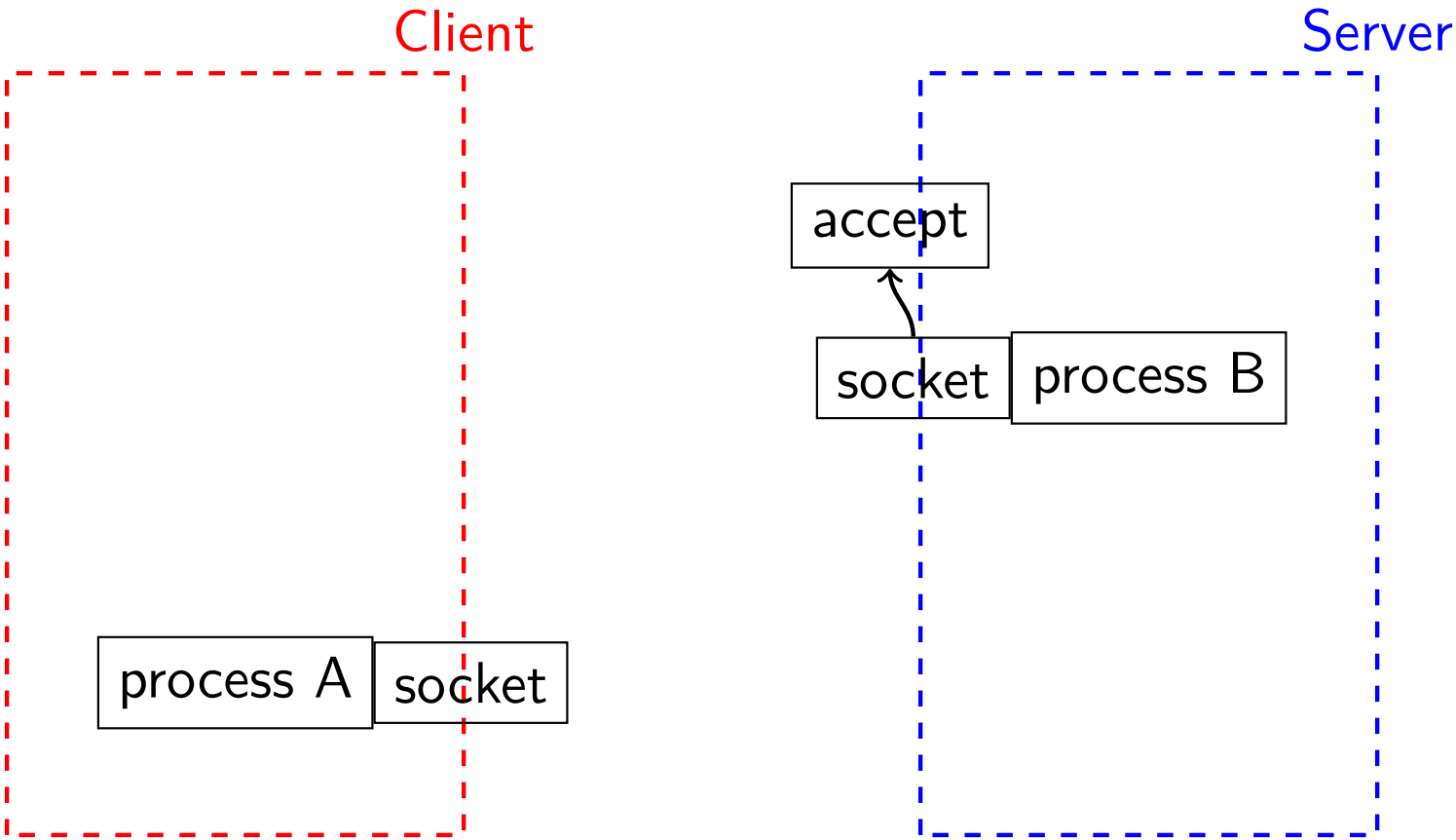
Socket 通信过程 I

服务器端的进程 process B 通过 socket 系统调用创建了一个 socket, 这个 socket 对外表现可以暂的理解为一个端口。比如 web 常用端口 80 。该 socket 随即调用 accept 方法, 等待客户端的连接。



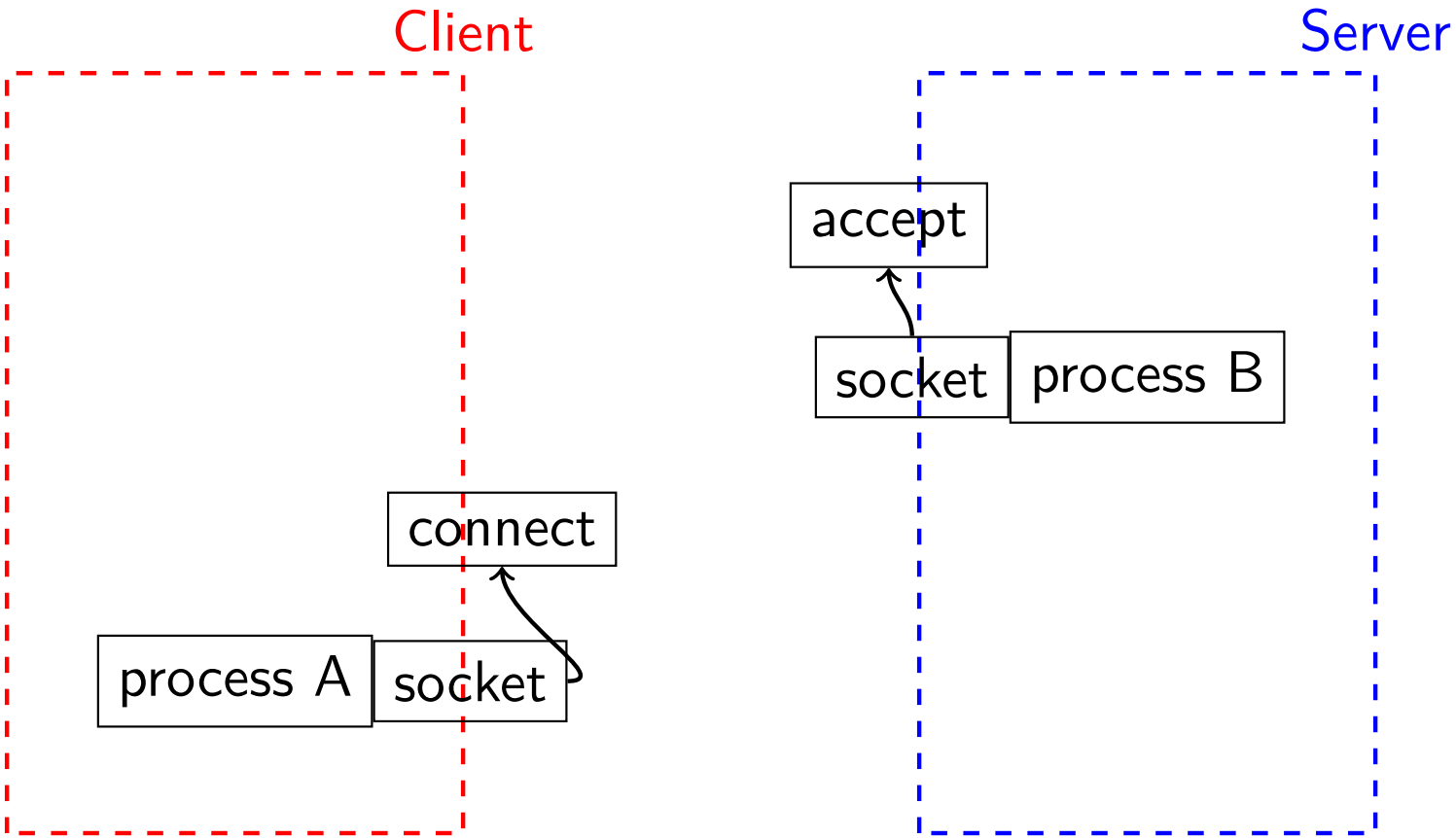
Socket 通信过程 II

客户端的进程 process A 通过 socket 系统调用创建一个 socket。



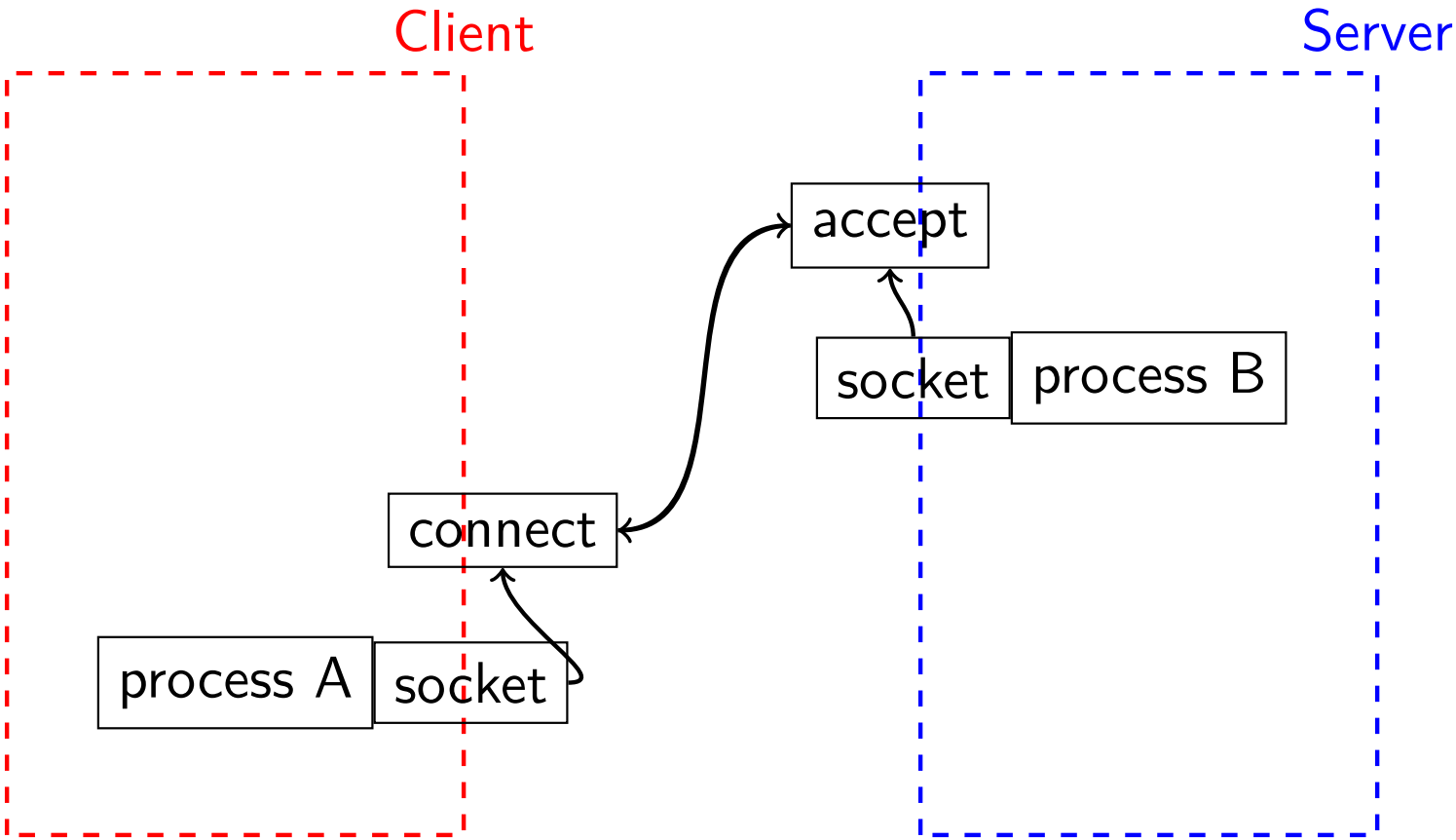
Socket 通信过程 III

客户端的进程 process A 创建的 socket 将服务器的地址和端口传递给 connect 方法。



Socket 通信过程 IV

客户端与服务端建立连接。然后 processA 和 B 就像操作文件描述符一样可以对 socket 进行操作了。



Create a Socket

socket 系统调用

socket() creates an endpoint for communication and returns a descriptor.

原型

```
#include <sys/types.h>
#include <sys/socket.h>

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

socket-domain

The domain parameter specifies a communications domain within which communication will take place; this selects the protocol family which should be used. These families are defined in the include file <sys/socket.h>. The domains are as table 1.

Name	**Purpose**
AF_UNIX, AF_LOCAL	Local communication
AF_INET	IPv4 Internet protocols
AF_INET6	IPv6 Internet protocols

Table: Domain of socket

socket-type

The socket has the indicated type, which specifies the semantics of communication. Currently defined types[1] are:

1. SOCK_STREAM
2. SOCK_DGRAM

A SOCK_STREAM type provides sequenced, reliable, connection based byte streams. An out-of-band data(like urgent mode in tcp) transmission mechanism may be supported.
A SOCK_DGRAM socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length).

[1] 一部分。

socket-protocol

The protocol used for communication is usually determined by the socket type and domain. There is normally no choice. The protocol parameter is used where there is a choice. 0 selects the default protocol, which is used in all the examples in our course.

Name a socket

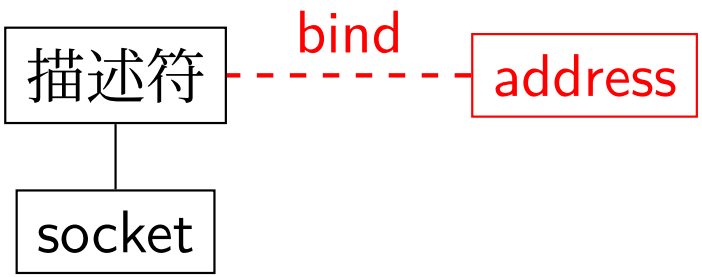
To make a socket (as created by a call to socket()) available for use by other processes, a server program needs to give the socket a name. Thus, AF_LOCAL sockets are associated with a file system pathname. AF_INET sockets are associated with an IP port number.

bind

bind 原型

```
#include <sys/types.h>
#include <sys/socket.h>

        int bind(int sockfd, const struct sockaddr *addr,
                 socklen_t addrlen);
```



On successful completion, bind returns 0. If it fails, it returns -1 and sets errno .

bind-addr

The actual structure passed for the addr argument will depend on the address family. The sockaddr structure is defined as something like:

```
struct sockaddr_in {
  sa_family_t    sin_family; /* address family: AF_INET */
  in_port_t      sin_port;   /* port in network byte order */
  struct in_addr sin_addr;   /* internet address */
};
/* Internet address. */
  struct in_addr {
  uint32_t       s_addr;     /* address in network byte order
};
```

The rules used in name binding vary between address families. Consult the manual entries in Section 7 for detailed information. For AF_INET see ip(7), for AF_INET6 see ipv6(7), for AF_UNIX see unix(7).

create a socket queue

To accept incoming connections on a socket, a server program must create a queue to store pending requests. It does this using the listen system call.

listen 原型

```
#include <sys/types.h>
#include <sys/socket.h>

      int listen(int sockfd, int backlog);
```

The listen function will return 0 on success or -1 on error.

accept connection

Once a server program has created and named a socket, it can wait for connections to be made to the socket by using the accept system call.

accept 原型

```
#include <sys/types.h>
#include <sys/socket.h>

 int accept(int sockfd, struct sockaddr *addr,
   socklen_t *addrlen);
```

The accept system call returns when a client program attempts to connect to the socket specified by the parameter socket.
The client is the first pending connection from that socket's queue.
The accept function creates a new socket to communicate with the client and returns its descriptor.
The new socket will have the same type as the server listen socket.

accept 补充

If there are no connections pending on the socket's queue, accept will block (so that the program won't continue) until a client does make connection. (We can change this behavior by using the O_NONBLOCK flag on the socket file descriptor, using the fcntl function.)

Requesting Connections

Client programs connect to servers by establishing a connection between an unnamed socket and the server listen socket. We can do this by calling connect.

connect 原型

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *addr,
                        socklen_t addrlen);
```

The socket specified by the parameter socket is connected to the server socket specified by the parameter address, which is of length address_len. The socket must be a valid file descriptor obtained by a call to socket. If it succeeds, connect returns 0, and -1 is returned on error.

close a socket

We can terminate a socket connection at the server and client by calling close, just as we would for low-level file descriptors.

close 原型

```
#include <unistd.h>

      int
      close(int fildes);
```

网络通信-客户端 I

引入头文件

```
cat client.c
#include<sys/types.h>
#include<sys/socket.h>
#include<stdio.h>
#include<stdlib.h>
#include<netinet/ip.h>
#include<netinet/in.h>
#include<unistd.h>
```

网络通信-客户端 II

创建 socket 连接

```
int main(){
 int sockfd, len, result;
//man 7 ip
 struct sockaddr_in address;
 char ch='C';
 sockfd=socket(AF_INET,SOCK_STREAM,0);
 address.sin_family=AF_INET;
 address.sin_addr.s_addr=inet_addr("127.0.0.1");
 address.sin_port=9980;
 len=sizeof(address);
 result=connect(sockfd,(struct sockaddr *)&address,len);
 if(result==-1){
  perror("oops:client!");
  exit(1);
 }
```

网络通信-客户端 III

对 socket 进行读写操作

```
cat client.c
int main(){
...
 write(sockfd,&ch,1);
 read(sockfd,&ch,1);
 printf("char from server is:%c\n",ch);
 close(sockfd);
 exit(0);
}
```

网络通信-服务端 I

创建一个 socket

```
cat server.c
int main(){
 int server_sockfd, client_sockfd;
 int server_len, client_len;
//man 7 ip
 struct sockaddr_in server_address;
 struct sockaddr_in client_address;
 server_sockfd=socket(AF_INET,SOCK_STREAM,0);
...
}
```

网络通信-服务端 II

bind

```
cat server.c
int main(){
...
server_sockfd=socket(AF_INET,SOCK_STREAM,0);
server_address.sin_family=AF_INET;
server_address.sin_addr.s_addr=inet_addr("127.0.0.1");
server_address.sin_port=9980;
server_len=sizeof(server_address);

bind(server_sockfd,
  (struct sockaddr *)&server_address,
  server_len);
...
}
```

网络通信-服务端 III

listen and accept

```
cat server.c
int main(){
...
listen(server_sockfd,5);
while(1){
 char ch;
 printf("serve...\n");
 client_len=sizeof(client_address);
 client_sockfd=accept(server_sockfd,
  (struct sockaddr *)&client_address,
  &client_len);
 read(client_sockfd,&ch,1);
 ch++;
 write(client_sockfd,&ch,1);
 close(client_sockfd);
}
```

查看监听的端口号

`server_address.sin_port=9980;` 当 server 运行起来时，通过 netstat 可能会发现本机监听的端口中并没有 9980, 这是因为通过 socket 传递的端口号是二进制数字，在各平台上的解析可能不一样。可做以下修改：

`server_address.sin_port=htons(9980);` [2]

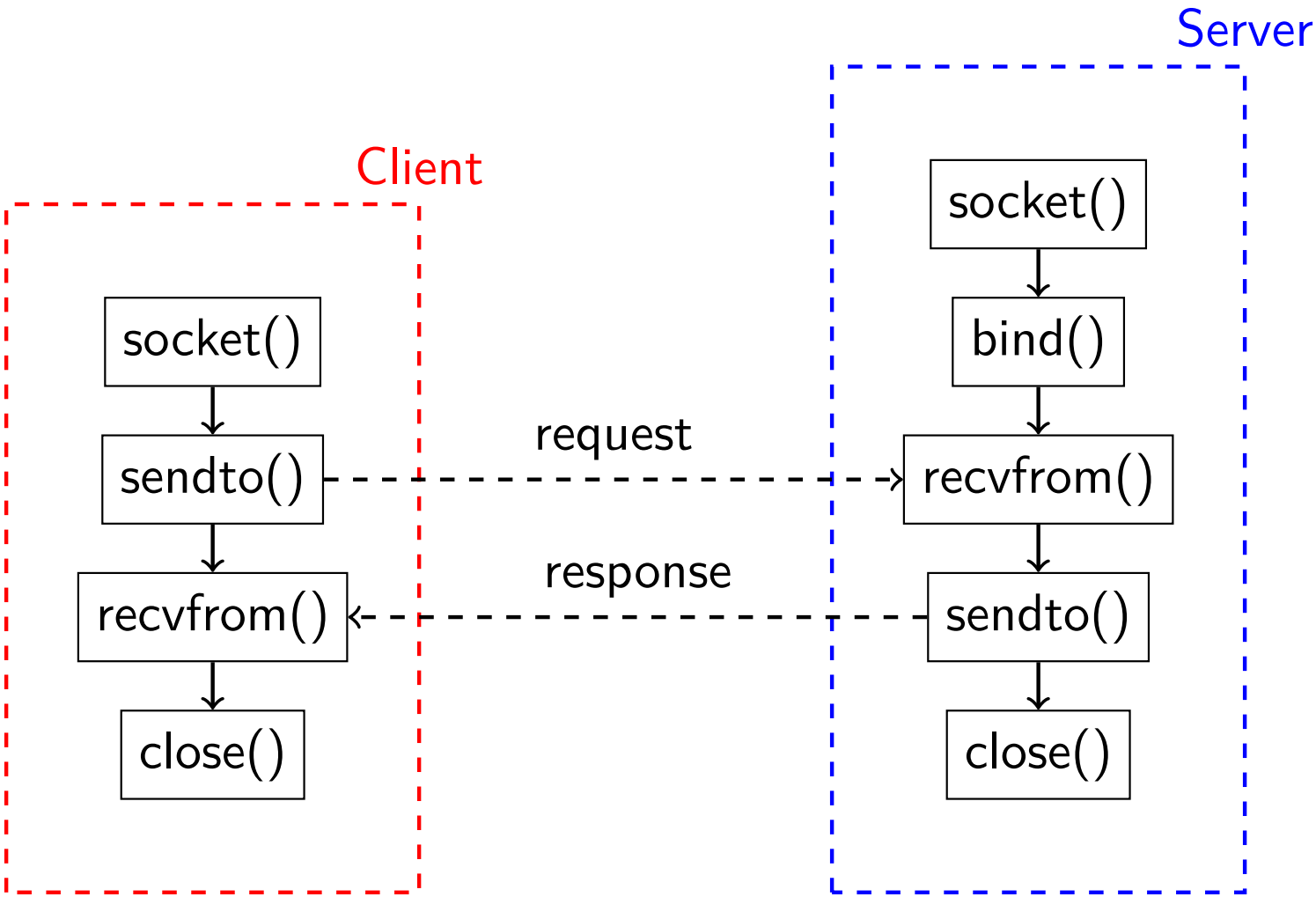
[2] 服务器端和客户端程序都要修改。

数据报

数据报

前面讲解的都是基于连接的通信方式（TCP）。当有些通信对数据的丢失不敏感或要求数据的质量不高的时候，可以考虑使用数据报来传递数据，对应到传输协议就是 UDP 协议。
编写 udp 服务程序，与 tcp 最明显的不同之处在于：需要使用两个数据报专用的系统调用 sendto 和 recvfrom 来代替 read 和 write 调用。

数据报通信过程


Server
Client
socket()
bind()
recvfrom()
sendto()
close()
socket()
sendto()
recvfrom()
close()
request
response

数据报例子-客户端

clientUDP.c

```
sockfd=socket(AF_INET,SOCK_DGRAM,0);
address.sin_family=AF_INET;
address.sin_addr.s_addr=inet_addr("127.0.0.1");
address.sin_port=htons(9980);
len=sizeof(address);

printf("input a char:");
buf=getchar();
int rz = sendto(sockfd,&buf,1,0,
  (struct sockaddr *)&address,len);
```

数据报例子-服务端

serverUDP.c

```
server_sockfd=socket(AF_INET,SOCK_DGRAM,0);
 server_address.sin_family=AF_INET;
 server_address.sin_addr.s_addr=inet_addr("127.0.0.1");
 server_address.sin_port=htons(9980);
 server_len=sizeof(server_address);
 bind(server_sockfd,
   (struct sockaddr *)&server_address,server_len);

while(1){
  char ch;
  printf("UDP serve...\n");
  recvfrom(server_sockfd,buf,1,0,
    (struct sockaddr *)&server_address,&server_len);
  printf("the client sent:%s",buf);
}
```

作业

编写一个 socket 程序，要求：

1. 使用 TCP 协议实现
2. 客户端可以和服务器端进行通信
3. 当用户输入 end 时，本客户端退出结束
4. 进阶要求：
   1. 多个客户端可以同时分别和服务端通信
   2. 实现一个类似聊天室的功能

The End

Appendix

本课程相关资源下载

1. ppt
   https://github.com/gmsft/ppt/tree/master/linux
2. 实验指导书
   https://github.com/gmsft/ppt/tree/master/book/linux

about man page

The manual is generally split into eight numbered sections, organized as follows (on Research Unix, BSD, macOS and Linux):

section	description
1	General commands
2	System calls
3	Library function(C standard library)
4	Special files(devices) and drivers
5	File formats and conventions
6	Games and screensavers
7	Miscellanea
8	System administration commands and daemons

Table: man page

在终端中运行 man read 与 man 2 read ， 观察其输出的区别。