## 视频内容

1. ICP\IP基础
2. socket编程
3. 进程间通信
4. 实战

## TCP\IP基础

1. OSI七层参考模型（理论标准） & TCP\IP四层模型（实际标准）

|  |  |
| --- | --- |
| 应用层 | http、ftp、pop3 |
| 表示层 | 数据压缩、加密等 |
| 会话层 | 管理会话 |
| 传输层 | tcp\udp，段 |
| 网络层 | Ip,icmp,igmp,报 |
| 数据链路层 | Frame 帧 |
| 物理层 | Byte |

OSI七层模型

|  |  |
| --- | --- |
| 应用层 |  |
| 传输层 |  |
| 网络层 |  |
| 链路层 | Arp、rarp |

TCP/IP四层模型

1. 概念：对等通信、封装、解封、端口

MTU:最大传输单元

路径MTU，传输路径上最小MTU

1. ping过程
   1. 根据域名到dns查询ip
   2. 在本地缓存查询ip对应的mac地址，若存在，直接在icmp中填充目的mac地址，然后发送icmp报文；否则，使用arp广播
   3. 符合目的ip的主机先将源目的ip和源mac地址存到自身缓存，填充arp报文的目的mac地址，返回给发送方
   4. 发送方收到回复，缓存目的ip的对应的mac，使用ping的icmp报文发送
2. IP头
3. TCP头
4. UDP头

## socket编程

### 地址结构

* 1. IPv4地址结构

struct sockaddr\_in **{**

uint8\_t sin\_len**;**

sa\_fimaly\_t sin\_family**;**

in\_port\_t sin\_port**;**

struct in\_addr sin\_addr**;**

char sin\_zone**[**8**];**

**}**

* 1. 通用地址结构

struct sockaddr**{**

uint8\_t sin\_len**;**

sa\_fimaly\_t sin\_family**;**

char sa\_data**[**14**];**

**}**

### 大小端转换

头文件<arpa/inet.h>

unsigned int i **=** 0x12345678**;**

char**\*** p **=** **reinterpret\_cast<**char**\*>(&**i**);**

printf**(**"%0x %0x %0x %0x\n"**,**p**[**0**],**p**[**1**],**p**[**2**],**p**[**3**]);**//输出78 56 34 12

i **=** htonl**(**i**);**

p **=** **reinterpret\_cast<**char**\*>(&**i**);**

printf**(**"%0x %0x %0x %0x\n"**,**p**[**0**],**p**[**1**],**p**[**2**],**p**[**3**]);**//输出12 34 56 78

### 点分法地址和网络地址转换函数

头文件包含<netinet/in.h>

**typedef** uint32\_t in\_addr\_t**;**

struct in\_addr **{**

in\_addr\_t s\_addr**;**

**};**

int inet\_aton**(**const char**\*** p**,** struct in\_addr **\*** inp**);**//0表示出错，1表示成功

char**\*** inet\_ntoa**(**truct in\_addr in**);**//返回点分ip

in\_addr\_t inet\_addr**(**const char **\***p**);**//返回uint32

char**\*** p **=** "192.168.188.146"**;**

struct in\_addr in**;**

**if(**inet\_aton**(**p**,&**in**)** **==** 0 **){**

printf**(**"%s\n"**,**"error"**);**

**}**

printf**(**"%u\n%u\n%s\n"**,**in**.**s\_addr**,**inet\_addr**(**p**),**inet\_ntoa**(**in**));**

### socket回显程序

* 1. 客户端：

#include<unistd.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<arpa/inet.h>

#include<netinet/in.h>

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<assert.h>

int main**(){**

int sock **=** 0**;**

assert**((**sock **=** socket**(**AF\_INET**,**SOCK\_STREAM**,**0**))** **!=** **-**1**);**

int n**;**

setsockopt**(**sock**,**SOL\_SOCKET**,**SO\_REUSEADDR**,&**n**,sizeof(**n**));**

struct sockaddr\_in addr\_srv**;**

addr\_srv**.**sin\_family **=** AF\_INET**;**

addr\_srv**.**sin\_port **=** htons**(**8000**);**

addr\_srv**.**sin\_addr**.**s\_addr **=** inet\_addr**(**"127.0.0.1"**);**

assert**(**connect**(**sock**,(**struct sockaddr**\*)&**addr\_srv**,sizeof(**addr\_srv**))** **!=** **-**1**);**

char revbuf**[**1024**]** **=** **{**0**};**

char sedbuf**[**1024**]** **=** **{**0**};**

int ret **=** 0**;**

**while((**fgets**(**sedbuf**,sizeof(**sedbuf**),**stdin**))** **!=** **NULL){**

write**(**sock**,**sedbuf**,**strlen**(**sedbuf**));**

assert**((**ret **=** read**(**sock**,&**revbuf**,sizeof(**revbuf**)))** **!=** **-**1**);**

**if(**ret **==** 0**){**

printf**(**"peer closed\n"**);**

**break;**

**}**

printf**(**"%s"**,**revbuf**);**

memset**(&**revbuf**,**0**,sizeof(**revbuf**));**

memset**(&**sedbuf**,**0**,sizeof(**sedbuf**));**

**}**

close**(**sock**);**

**return** 0**;**

**}**

* 1. 服务端：（可接受多客户端连接）

#include<unistd.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<arpa/inet.h>

#include<netinet/in.h>

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<assert.h>

void do\_server**(**int conn**){**

char revbuf**[**1024**]** **=** **{**0**};**

int ret **=** 0**;**

**while(**1**){**

memset**(&**revbuf**,**0**,sizeof(**revbuf**));**

assert**((**ret **=** read**(**conn**,**revbuf**,sizeof(**revbuf**)))** **!=** **-**1**);**

**if(**ret **==** 0**){**

printf**(**"peer closed\n"**);**

**break;**

**}**

printf**(**"%s"**,**revbuf**);**

write**(**conn**,**revbuf**,**strlen**(**revbuf**));**

**}**

**}**

int main**(){**

int fd\_lst **=** 0**;**

assert**((**fd\_lst **=** socket**(**AF\_INET**,**SOCK\_STREAM**,**0**))** **!=** **-**1**);**

int n**;**

setsockopt**(**fd\_lst**,**SOL\_SOCKET**,**SO\_REUSEADDR**,&**n**,sizeof(**n**));**

struct sockaddr\_in addr\_srv**;**

memset**(&**addr\_srv**,**0**,sizeof(**addr\_srv**));**

addr\_srv**.**sin\_family **=** AF\_INET**;**

addr\_srv**.**sin\_port **=** htons**(**8000**);**

addr\_srv**.**sin\_addr**.**s\_addr **=** inet\_addr**(**"127.0.0.1"**);**

assert**(**bind**(**fd\_lst**,(**struct sockaddr**\*)&**addr\_srv**,sizeof(**addr\_srv**))** **!=** **-**1**);**

assert**(**listen**(**fd\_lst**,**SOMAXCONN**)** **!=** **-**1**);**

int conn **=** 0**;**

struct sockaddr\_in addr\_cli**;**

socklen\_t len **=** **sizeof(**addr\_cli**);**

**while(**1**){**

assert**((**conn **=** accept**(**fd\_lst**,(**struct sockaddr**\*)&**addr\_cli**,&**len**))** **!=** **-**1 **);**

printf**(**"clinet ip: %s, port: %d\n"**,**inet\_ntoa**(**addr\_cli**.**sin\_addr**),**ntohs**(**addr\_cli**.**sin\_port**));**

pid\_t pid**;**

assert**((**pid **=** fork**())** **!=** **-**1**);**

**if(**pid **==** 0**){**

close**(**fd\_lst**);**

do\_server**(**conn**);**

close**(**conn**);**

**break;**

**}**

**else{**

close**(**conn**);**

**continue;**

**}**

**}**

**return** 0**;**

**}**

### 聊天程序（处理粘包问题）：

* 1. 服务端：//会出现父进程死亡，子进程未退出的情况，使用信号的方式关闭子进程

#include <unistd.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <sys/prctl.h>

#include <signal.h>

#include <stdio.h>

#include <string.h>

#include <assert.h>

int static st\_sig **=** 0**;**

struct Packet**{**

int len**;**

char buf**[**1024**];**

**};**

void do\_sed**(**int conn**){**

Packet sedbuf**;**

memset**(&**sedbuf**,**0**,sizeof(**sedbuf**));**

**while(**fgets**(**sedbuf**.**buf**,sizeof(**sedbuf**.**buf**),**stdin**)** **!=** **NULL){**

int n **=** strlen**(**sedbuf**.**buf**);**

sedbuf**.**len **=** htons**(**n**);**

write**(**conn**,&**sedbuf**,**n**+**4**);**

memset**(&**sedbuf**,**0**,sizeof(**sedbuf**));**

**}**

**}**

void do\_rev**(**int conn**){**

Packet revbuf**;**

**while(**1**){**

memset**(&**revbuf**,**0**,sizeof(**revbuf**));**

int n **=** 0**;**

int ret **=** read**(**conn**,&**n**,**4**);**

assert**(**ret **!=** **-**1**);**

**if(**ret **==** 0**){**

printf**(**"peer closed\n"**);**

**break;**

**}**

revbuf**.**len **=** ntohs**(**n**);**

ret **=** read**(**conn**,&**revbuf**.**buf**,**revbuf**.**len**);**

assert**(**ret **!=** **-**1**);**

**if(**ret **==** 0**){**

printf**(**"peer closed\n"**);**

**break;**

**}**

printf**(**"%s"**,**revbuf**.**buf**);**

**}**

**}**

void handle**(**int sig**){**

**if(**sig **==** SIGHUP**){**

printf**(**"child received peer closed\n"**);**

st\_sig **=** 1**;**

**}**

**}**

int main**(){**

int fd\_lst **=** 0**;**

assert**((**fd\_lst **=** socket**(**AF\_INET**,**SOCK\_STREAM**,**0**))** **!=** **-**1**);**

int n**;**

setsockopt**(**fd\_lst**,**SOL\_SOCKET**,**SO\_REUSEADDR**,&**n**,sizeof(**n**));**

struct sockaddr\_in srv\_addr**;**

srv\_addr**.**sin\_family **=** AF\_INET**;**

srv\_addr**.**sin\_port **=** htons**(**8000**);**

srv\_addr**.**sin\_addr**.**s\_addr **=** inet\_addr**(**"127.0.0.1"**);**

assert**((**bind**(**fd\_lst**,(**struct sockaddr**\*)&**srv\_addr**,sizeof(**srv\_addr**)))** **!=** **-**1**);**

assert**((**listen**(**fd\_lst**,**SOMAXCONN**))** **!=** **-**1**);**

int conn **=** 0**;**

sockaddr\_in cli\_addr**;**

socklen\_t len **=** **sizeof(**cli\_addr**);**

assert**((**conn **=** accept**(**fd\_lst**,(**struct sockaddr**\*)&**cli\_addr**,&**len**))** **!=** **-**1**);**

printf**(**"Connect from %s : %d\n"**,**inet\_ntoa**(**cli\_addr**.**sin\_addr**),**ntohs**(**cli\_addr**.**sin\_port**));**

pid\_t pid **=** 0 **;**

assert**((**pid **=** fork**())** **!=** **-**1**);**

**if(**pid **==** 0**){**

signal**(**SIGHUP**,**handle**);**

prctl**(**PR\_SET\_PDEATHSIG**,**SIGHUP**);**

sleep**(**1**);**

**if(**st\_sig**){**

printf**(**"child closed\n"**);**

**return** 0**;**

**}**

do\_sed**(**conn**);**

**}**

**else{**

do\_rev**(**conn**);**

**}**

close**(**conn**);**

close**(**fd\_lst**);**

**return** 0**;**

**}**

* 1. 客户端：

#include <unistd.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <signal.h>

#include <sys/prctl.h>

#include <stdio.h>

#include <string.h>

#include <assert.h>

#include <stdlib.h>

struct Packet**{**

int len**;**

char buf**[**1024**];**

**};**

void do\_sed**(**int conn**){**

Packet sedbuf**;**

memset**(&**sedbuf**,**0**,sizeof(**sedbuf**));**

**while(**fgets**(**sedbuf**.**buf**,sizeof(**sedbuf**.**buf**),**stdin**)** **!=** **NULL){**

int n **=** strlen**(**sedbuf**.**buf**);**

sedbuf**.**len **=** htons**(**n**);**

write**(**conn**,&**sedbuf**,**n**+**4**);**

memset**(&**sedbuf**,**0**,sizeof(**sedbuf**));**

**}**

**}**

void do\_rev**(**int conn**){**

Packet revbuf**;**

**while(**1**){**

memset**(&**revbuf**,**0**,sizeof(**revbuf**));**

int n **=** 0**;**

int ret **=** read**(**conn**,&**n**,**4**);**

assert**(**ret **!=** **-**1**);**

**if(**ret **==** 0**){**

printf**(**"peer closed\n"**);**

**break;**

**}**

revbuf**.**len **=** ntohs**(**n**);**

ret **=** read**(**conn**,**revbuf**.**buf**,**revbuf**.**len**);**

assert**(**ret **!=** **-**1**);**

**if(**ret **==** 0**){**

printf**(**"peer closed\n"**);**

**break;**

**}**

printf**(**"%s"**,**revbuf**.**buf**);**

**}**

**}**

void handle**(**int sig**){**

**if(**sig **==** SIGHUP**){**

printf**(**"handle \n"**);**

exit**(**0**);**

**}**

**}**

int main**(){**

int sock **=** 0**;**

assert**((**sock **=** socket**(**AF\_INET**,**SOCK\_STREAM**,**0**))** **!=** **-**1**);**

struct sockaddr\_in srv\_addr**;**

srv\_addr**.**sin\_family **=** AF\_INET**;**

srv\_addr**.**sin\_port **=** htons**(**8000**);**

srv\_addr**.**sin\_addr**.**s\_addr **=** inet\_addr**(**"127.0.0.1"**);**

assert**(**connect**(**sock**,(**struct sockaddr**\*)&**srv\_addr**,sizeof(**srv\_addr**))** **!=** **-**1**);**

pid\_t pid **=** 0 **;**

assert**((**pid **=** fork**())** **!=** **-**1**);**

**if(**pid **==** 0**){**

signal**(**SIGHUP**,**handle**);**

prctl**(**PR\_SET\_PDEATHSIG**,**SIGHUP**);**

do\_sed**(**sock**);**

**}**

**else{**

do\_rev**(**sock**);**

**}**

close**(**sock**);**

**return** 0**;**

**}**

1. gethostname() gethostbyname()

#include <unistd.h>

#include <netdb.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <string.h>

#include <assert.h>

int main**(){**

char name**[**1024**];**

assert**(**gethostname**(**name**,sizeof(**name**))** **!=** **-**1**);**

printf**(**"%s\n"**,**name**);**

struct hostent**\*** he**;**

he **=** gethostbyname**(**name**);**

int len **=** he**->**h\_length**;** //并不代表有几个ip，代表ip总共占用的字节数

int i **=** 0 **;**

**while(**he**->**h\_addr\_list**[**i**]** **!=** **NULL){**

printf**(**"%s\n"**,**inet\_ntoa**(\*(**struct in\_addr**\*)**he**->**h\_addr\_list**[**i**]));**//必须使用 inet\_ntoa 将网络地址转化

**++**i**;**

**}**

**return** 0**;**

**}**

### tcp的11种状态

* 1. SYN\_SEND
  2. LISTEN
  3. SYN\_RECD
  4. ESTIBLISHED
  5. FIN\_WAIT\_1
  6. CLOSED\_WAIT
  7. FIN\_WAIT\_2
  8. LAST\_ACK
  9. TIME\_WAIT
  10. CLOSED
  11. CLOSING(特殊情况下出现，当客户端和服务端同时发送FIN信号，不会出现FIN\_WAIT\_1,FIN\_WAIT\_2状态，取而代之的是CLOSING状态)

### SIGPIPE

当对接方关闭的情况下，发送信息，对接方会恢复RST，在接收到RST后，还继续给对接方发送信息，就会收到对接方返回的SIGPIPE信号

### 五种I/O模型

1. 阻塞
2. 非阻塞
3. I/O复用
4. 信号驱动
5. 异步I/O

信号驱动和异步I/O区别是，信号驱动是用户进程主动拉取数据，异步I/O是内核推送数据到用户进程区间

### select

|  |
| --- |
| int select**(**int nfd**,**fd\_set**\*** read\_fds**,** fd\_set**\*** write\_fds **,**  fd\_set**\*** except\_fds**,**struct timeval**\*** timeout**)**  返回可操作的文件数目 |
| FD\_ZERO**(**fd\_set**\*** fds**);**  FD\_SET**(**int fd**,**fd\_set**\*** fds**);**  FD\_ISSET**(**int fd**,**fd\_set**\*** fds**);**  FD\_CLR**(**int fd**,**fd\_set**\*** fds**);** |

1 #include**<**unistd**.**h**>**

2 #include**<**sys**/**types**.**h**>**

3 #include**<**sys**/**socket**.**h**>**

4 #include**<**netinet**/**in**.**h**>**

5 #include**<**arpa**/**inet**.**h**>**

6 #include**<**sys**/**time**.**h**>**

7 #include**<**errno**.**h**>**

8 #include**<**signal**.**h**>**

9

10 #include**<**stdio**.**h**>**

11 #include**<**string**.**h**>**

12 #include**<**assert**.**h**>**

13

14 void handle**(**int sig**){**

15 printf**(**"sig %d\n"**,**sig**);**

16 **}**

17 int main**(){**

18

19 signal**(**SIGPIPE**,**handle**);**

20 int sock **=** 0**;**

21 assert**((**sock **=** socket**(**AF\_INET**,**SOCK\_STREAM**,**0**))** **!=** **-**1**);**

22 struct sockaddr\_in addr**;**

23 addr**.**sin\_family **=** AF\_INET**;**

24 addr**.**sin\_port **=** htons**(**8000**);**

25 int n **=** 0**;**

26 setsockopt**(**sock**,**SOL\_SOCKET**,**SO\_REUSEADDR**,&**n**,sizeof(**n**));**

27 addr**.**sin\_addr**.**s\_addr **=** inet\_addr**(**"127.0.0.1"**);**

28 assert**(**connect**(**sock**,(**struct sockaddr**\*)&**addr**,sizeof(**addr**))** **!=** **-**1**);**

29 char revbuf**[**1024**]** **=** **{**0**}** **;**

30 char sedbuf**[**1024**]** **=** **{**0**};**

31 fd\_set read\_set**;**

32 FD\_ZERO**(&**read\_set**);**

33 int in\_no **=** fileno**(**stdin**);**

34 int maxfd **=** sock**+**1**;**

35 **while(**1**){**

36 FD\_SET**(**in\_no**,&**read\_set**);**//注意：这两句必须写在while循环内

37 FD\_SET**(**sock**,&**read\_set**);**

38 int ret **=** select**(**maxfd**,&**read\_set**,NULL,NULL,NULL);**//在此处有数据阻塞，直到感兴趣的fd有数据可读

39 **if(**ret **<=** 0**){**

40 printf**(**"select error\n"**);**

41 **return** 0**;**

42 **}**

43 **if(**FD\_ISSET**(**sock**,&**read\_set**)){**

44 int ret **=** read**(**sock**,**revbuf**,sizeof(**revbuf**));**

45 **if(**ret **==** **-**1 **&&** errno **==** EINTR**){**

46 **continue;**

47 **}**

48 **if(**ret **==** **-**1**){**

49 printf**(**"error\n"**);**

50 **break;**

51 **}**

52 **if(**ret **==** 0**){**

53 printf**(**"peer closed\n"**);**

54 char s**[**1024**]** **=** "11111"**;**

55 write**(**sock**,**s**,**strlen**(**s**));**//当对等方关闭，sock会不阻塞，read返回0，该句执行后，srv回应RST

56 write**(**sock**,**s**,**strlen**(**s**));**//收到RST后，继续发送数据，收到SIGPIPE

57 **break;**

58 **}**

59 printf**(**"%s"**,**revbuf**);**

60 **}**

61 **if(**FD\_ISSET**(**in\_no**,&**read\_set**)){**

62 fgets**(**sedbuf**,sizeof(**sedbuf**),**stdin**);**

63 write**(**sock**,**sedbuf**,**strlen**(**sedbuf**));**

64 **}**

65 memset**(**sedbuf**,**0**,sizeof(**sedbuf**));**

66 memset**(**revbuf**,**0**,sizeof(**revbuf**));**

67 **}**

68 close**(**sock**);**

69 **return** 0**;**

70 **}**

int shutdown(int sockfd, int how);成功返回0，失败返回-1；how取值：SHUT\_RD,SHUT\_WR,SHUT\_RDWR

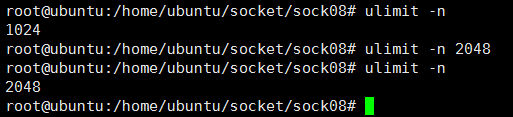
* 1. 可以选择关闭写或者读或者读写
  2. 和引用计数无关，close只是引用计数减一，为0时关闭

### poll

1. select缺陷和poll的使用
   1. select缺点：
      1. 并发数受进程能打开最大文件数目限制

使用ulimit –u查询进程最大打开文件数

ulimit –u N改变当前进程最大打开文件数



通过程序改变进程所能打开的最大文件数：

27 /\*

28 getrlimit(int resource,struct rlimit\* rlim);

29 setrlimit(int resource,const struct limit\* rlim)

30 struct rlimit{

31 rlim\_t rlim\_cur;

32 rlim\_t rlim\_max;

33 }

34 resource = RLIMIT\_NOFILE

35 \*/

36 struct rlimit rlim**;**

37 assert**((**getrlimit**(**RLIMIT\_NOFILE**,&**rlim**))!=-**1**);**

38 printf**(**"%d, %d\n"**,(**int**)**rlim**.**rlim\_cur**,(**int**)**rlim**.**rlim\_max**);**

39 rlim**.**rlim\_cur **=** 3072**;**

40 rlim**.**rlim\_max **=** 8088**;**

41 assert**((**setrlimit**(**RLIMIT\_NOFILE**,&**rlim**))** **!=** **-**1**);**

42 printf**(**"%d, %d\n"**,(**int**)**rlim**.**rlim\_cur**,(**int**)**rlim**.**rlim\_max**);**

**输出：**

**1024，2048**

**3072，8088**

* + 1. 并发数受fd\_set容量限制，FD\_SETSIZE

只能通过编译内核改变，宏变量

实验：已将进程所能打开的最大文件数改为了>1024

结果：服务端能处理的连接数数1020（因为0（stdin），1（stdout），2（stderr），3（fd\_lst）已经打开）；客户端能连接1021（较服务端少监听fd\_lst）

* 1. poll使用

服务端：

27 int count **=** 0**;**

28 struct pollfd fds**[**2048**]** **=** **{**0**};**

29 **for(**int i **=** 0 **;** i **<** 2048**;++**i**){**

30 fds**[**i**].**fd **=** **-**1**;**

31 **}**

32 fds**[**0**].**fd **=** fd\_lst**;**

33 fds**[**0**].**events **=** POLLIN **;**

34 int maxi **=** 1**;**

35 **while(**1**){**

36 assert**((**poll**(**fds**,**maxi**,-**1**))>**0**);**

37 memset**(&**cliaddr**,**0**,sizeof(**cliaddr**));**

38 assert**((**conn **=** accept**(**fd\_lst**,(**struct sockaddr**\*)&**cliaddr**,&**len**))** **!=** **-**1**);**

39 printf**(**"count = %d , addr = %s , port = %d\n"**,++**count**,**inet\_ntoa**(**cliaddr**.**sin\_addr**),**ntohs**(**cliaddr**.**sin\_port**)** **);**

40 **}**

最大可连接数大于1024

### epoll

|  |
| --- |
| int epoll\_create**(**int size**);**//使用hash  int epoll\_create1**(**int flags**);**//使用红黑树  int epoll\_ctl**(**int epollfd**,**EPOLL\_CTL\_ADD**,**int sock**,**epoll\_event**\*** ev**);**  int epoll\_ctl**(**int epollfd**,**EPOLL\_CTL\_DEL**,**int sock**,**epoll\_event**\*** ev**);**  epoll\_wait**(**int epollfd**,**epoll\_event**\*** events**,**int maxevents**,**int timeout**);** |
| 服务端：  1 #include**<**unistd**.**h**>**  2 #include**<**sys**/**socket**.**h**>**  3 #include**<**sys**/**types**.**h**>**  4 #include**<**netinet**/**in**.**h**>**  5 #include**<**arpa**/**inet**.**h**>**  6 #include**<**errno**.**h**>**  7 #include**<**fcntl**.**h**>**  8 #include**<**sys**/**epoll**.**h**>**  9  10 #include**<**stdio**.**h**>**  11 #include**<**assert**.**h**>**  12 #include**<**string**.**h**>**  13 #include**<**stdlib**.**h**>**  14  15 void setnoblocking**(**int conn**){**  16 int ret **=** fcntl**(**conn**,**F\_SETFL**,**fcntl**(**conn**,**F\_GETFL**)|**O\_NONBLOCK**);**  17 assert**(**ret **!=** **-**1**);**  18 **}**  19 void do\_echo**(**int efd**,**struct epoll\_event ev**){**  20 char revbuf**[**1024**]** **=** **{**0**};**  21 **while(**1**){**  22 int ret **=** read**(**ev**.**data**.**fd**,**revbuf**,sizeof(**revbuf**));**  23 **if(**ret **==** **-**1**){**  24 **if(**errno **==** EINTR**)**  25 **continue;**  26 **break;**  27 **}**  28 **if(**ret **==** 0**){**  29 printf**(**"client %d closed\n"**,**ev**.**data**.**fd**);**  30 epoll\_ctl**(**efd**,**EPOLL\_CTL\_DEL**,**ev**.**data**.**fd**,&**ev**);**  31 **break;**  32 **}**  33 fputs**(**revbuf**,**stdout**);**  34 write**(**ev**.**data**.**fd**,**revbuf**,**strlen**(**revbuf**));**  35 memset**(**revbuf**,**0**,sizeof(**revbuf**));**  36 **}**  37 **}**  38 int main**(){**  39 int lis **=** 0**;**  40 assert**((**lis **=** socket**(**AF\_INET**,**SOCK\_STREAM**,**0**))** **!=** **-**1**);**  41 int n **=** 1**;**  42 setsockopt**(**lis**,**SOL\_SOCKET**,**SO\_REUSEADDR**,&**n**,sizeof(**n**));**  43 struct sockaddr\_in srvaddr**;**  44 memset**(&**srvaddr**,**0**,sizeof(**srvaddr**));**  45 srvaddr**.**sin\_family **=** AF\_INET**;**  46 srvaddr**.**sin\_port **=** htons**(**8000**);**  47 srvaddr**.**sin\_addr**.**s\_addr **=** inet\_addr**(**"127.0.0.1"**);**  48 assert**(**bind**(**lis**,(**struct sockaddr**\*)&**srvaddr**,sizeof(**srvaddr**))** **!=** **-**1**);**  49 assert**(**listen**(**lis**,**SOMAXCONN**)** **!=** **-**1**);**  50 int conn**;**  51 struct sockaddr\_in cliaddr**;**  52 memset**(&**cliaddr**,**0**,sizeof(**cliaddr**));**  53 socklen\_t len **=** **sizeof(**cliaddr**);**  54 int efd **=** epoll\_create1**(**0**);**  55 assert**(**efd **!=** **-**1**);**  56 struct epoll\_event events**[**2048**]** **=** **{**0**};**  57 **for(**int i **=** 0 **;** i **<** 2048**;++**i**){**  58 events**[**i**].**data**.**fd **=** **-**1**;**  59 **}**  60 struct epoll\_event ev**;**  61 ev**.**data**.**fd **=** lis**;**  62 ev**.**events **=** EPOLLIN**;**  63 assert**(**epoll\_ctl**(**efd**,**EPOLL\_CTL\_ADD**,**lis**,&**ev**)** **!=** **-**1**);**  64 int nfd **=** 0**;**  65 **while(**1**){**  66 nfd **=** epoll\_wait**(**efd**,**events**,**2048**,-**1**);**  67 assert**(**nfd **!=** **-**1**);**  68 **if(**nfd **==** 0**){**  69 printf**(**"epoll\_wait error\n"**);**  70 exit**(-**1**);**  71 **}**  72 **for(**int i **=** 0 **;**i **<** nfd **;** **++**i**){**  73 **if(**events**[**i**].**data**.**fd **==** **-**1**)**  74 **continue;**  75 **if(**events**[**i**].**data**.**fd **==** lis**){**  76 assert**((**conn **=** accept**(**lis**,(**struct sockaddr**\*)&**cliaddr**,&**len**))** **!=** **-**1**);**  77 printf**(**"addr = %s , port = %d\n"**,**inet\_ntoa**(**cliaddr**.**sin\_addr**),**ntohs**(**cliaddr**.**sin\_port**));**  78 setnoblocking**(**conn**);**  79 ev**.**data**.**fd **=** conn**;**  80 ev**.**events **=** EPOLLIN **|** EPOLLET**;**  81 assert**(**epoll\_ctl**(**efd**,**EPOLL\_CTL\_ADD**,**conn**,&**ev**)** **!=** **-**1**);**  82 **}**  83 **else{**  84 do\_echo**(**efd**,**events**[**i**]);**  85 **}**  86 **}**  87 **}**  88 **return** 0**;**  89 **}** |
| 客户端：  1 #include**<**unistd**.**h**>**  2 #include**<**sys**/**socket**.**h**>**  3 #include**<**sys**/**types**.**h**>**  4 #include**<**netinet**/**in**.**h**>**  5 #include**<**arpa**/**inet**.**h**>**  6 #include**<**errno**.**h**>**  7  8 #include**<**stdio**.**h**>**  9 #include**<**assert**.**h**>**  10 #include**<**string**.**h**>**  11 #include**<**stdlib**.**h**>**  12  13 int main**(){**  14 int sock **=** 0**;**  15 assert**((**sock **=** socket**(**AF\_INET**,**SOCK\_STREAM**,**0**))** **!=** **-**1**);**  16 struct sockaddr\_in srvaddr**;**  17 memset**(&**srvaddr**,**0**,sizeof(**srvaddr**));**  18 srvaddr**.**sin\_family **=** AF\_INET**;**  19 srvaddr**.**sin\_port **=** htons**(**8000**);**  20 srvaddr**.**sin\_addr**.**s\_addr **=** inet\_addr**(**"127.0.0.1"**);**  21 assert**(**connect**(**sock**,(**struct sockaddr**\*)&**srvaddr**,sizeof(**srvaddr**))** **!=** **-**1**);**  22 char revbuf**[**1024**]** **=** **{**0**};**  23 char sedbuf**[**1024**]** **=** **{**0**};**  24 **while(**fgets**(**sedbuf**,sizeof(**sedbuf**),**stdin**)** **!=** **NULL){**  25 write**(**sock**,**sedbuf**,**strlen**(**sedbuf**));**  26 int ret **=** read**(**sock**,**revbuf**,sizeof(**revbuf**));**  27 **if(**ret **==** **-**1**){**  28 **if(**errno **==** EINTR**){**  29 **continue;**  30 **}**  31 exit**(-**1**);**  32 **}**  33 fputs**(**revbuf**,**stdout**);**  34 memset**(**sedbuf**,**0**,sizeof(**sedbuf**));**  35 memset**(**revbuf**,**0**,sizeof(**revbuf**));**  36 **}**  37 **return** 0**;**  38 **}** |

### UDP编程

1. UDP特点：
   * 1. 非连接
     2. 不可靠
     3. 面向消息的，tcp面向字节，所以udp没有粘包问题
     4. 一般情况下比较高效
2. udp编程模型：
   * 1. 服务端：socket、bind、不需要listen和accept
     2. 客户端：socket，connect不是必须的
3. udp回射服务端客户端

|  |
| --- |
| 服务端：  1 #include**<**unistd**.**h**>**  2 #include**<**sys**/**types**.**h**>**  3 #include**<**sys**/**socket**.**h**>**  4 #include**<**netinet**/**in**.**h**>**  5 #include**<**arpa**/**inet**.**h**>**  6 #include**<**errno**.**h**>**  7  8 #include**<**stdio**.**h**>**  9 #include**<**stdlib**.**h**>**  10 #include**<**assert**.**h**>**  11 #include**<**string**.**h**>**  12  13 int main**(){**  14 int listen**;**  15 assert**((**listen **=** socket**(**AF\_INET**,**SOCK\_DGRAM**,**0**))** **!=** **-**1**);**  16 struct sockaddr\_in srvaddr**;**  17 memset**(&**srvaddr**,**0**,sizeof(**srvaddr**));**  18 srvaddr**.**sin\_family **=** AF\_INET**;**  19 srvaddr**.**sin\_port **=** htons**(**8000**);**  20 srvaddr**.**sin\_addr**.**s\_addr **=** inet\_addr**(**"127.0.0.1"**);**  21 assert**((**bind**(**listen**,(**struct sockaddr**\*)&**srvaddr**,sizeof(**srvaddr**)))** **!=** **-**1**);**  22 struct sockaddr\_in cliaddr**;**  23 memset**(&**cliaddr**,**0**,sizeof(**cliaddr**));**  24 socklen\_t len **=** **sizeof(**cliaddr**);**  25 char revbuf**[**1024**]** **=** **{**0**};**  26 int ret **=** 0**;**  27 **while(**1**){**  28 memset**(**revbuf**,**0**,sizeof(**revbuf**));**  29 ret **=** recvfrom**(**listen**,**revbuf**,sizeof(**revbuf**),**0**,(**struct sockaddr**\*)&**cliaddr**,&**len**);**  30 **if(**ret **==** **-**1**){**  31 **if(**errno **==** EINTR**){**  32 **continue;**  33 **}**  34 **else{**  35 printf**(**"revbuf return -1\n"**);**  36 exit**(-**1**);**  37 **}**  38 **}**  39 fputs**(**revbuf**,**stdout**);**  40 ret **=** sendto**(**listen**,**revbuf**,**strlen**(**revbuf**),**0**,(**struct sockaddr**\*)&**cliaddr**,**len**);**  41 **if(**ret **==** **-**1**){**  42 **if(**errno **==** EINTR**){**  43 **continue;**  44 **}**  45 **else{**  46 printf**(**"sendto return -1\n"**);**  47 exit**(-**1**);**  48 **}**  49 **}**  50 **}**  51 **return** 0**;**  52 **}** |
| 客户端：  1 #include**<**unistd**.**h**>**  2 #include**<**sys**/**types**.**h**>**  3 #include**<**sys**/**socket**.**h**>**  4 #include**<**netinet**/**in**.**h**>**  5 #include**<**arpa**/**inet**.**h**>**  6 #include**<**errno**.**h**>**  7  8 #include**<**stdio**.**h**>**  9 #include**<**stdlib**.**h**>**  7  8 #include**<**stdio**.**h**>**  9 #include**<**stdlib**.**h**>**  10 #include**<**assert**.**h**>**  11 #include**<**string**.**h**>**  12  13 int main**(){**  14 int sock**;**  15 assert**((**sock **=** socket**(**AF\_INET**,**SOCK\_DGRAM**,**0**))** **!=** **-**1**);**  16 struct sockaddr\_in srvaddr**;**  17 memset**(&**srvaddr**,**0**,sizeof(**srvaddr**));**  18 srvaddr**.**sin\_family **=** AF\_INET**;**  19 srvaddr**.**sin\_port **=** htons**(**8000**);**  20 srvaddr**.**sin\_addr**.**s\_addr **=** inet\_addr**(**"127.0.0.1"**);**  21 char sedbuf**[**1024**]** **=** **{**0**};**  22 char revbuf**[**1024**]** **=** **{**0**};**  23 **while(**fgets**(**sedbuf**,sizeof(**sedbuf**),**stdin**)!=NULL){**  24 sendto**(**sock**,**sedbuf**,**strlen**(**sedbuf**),**0**,(**struct sockaddr**\*)&**srvaddr**,sizeof(**srvaddr**));**  25 recvfrom**(**sock**,**revbuf**,sizeof(**revbuf**),**0**,NULL,NULL);**  26 fputs**(**revbuf**,**stdout**);**  27 memset**(**sedbuf**,**0**,sizeof(**sedbuf**));**  28 memset**(**revbuf**,**0**,sizeof(**revbuf**));**  29 **}**  30 **return** 0**;**  31 **}** |

1. udp注意问题
2. 有可能丢包、重复、无序
3. 没有流量控制机制（tcp滑动窗口），服务端缓冲区有可能被覆盖
4. 有可能被截断。udp面向消息的，如果revbuf小于消息长度，则截断，缓冲区不会保留数据
5. recvfrom返回0并不代表对等方关闭，而是对等放发送了长度为0的消息
6. icmp异步错。当服务端没有开启时，客户端进行发送数据，回收到icmp错，但是无连接不能接受异步错，所以客户端一直阻塞在recvfrom，而且不能得到icmp异步错
7. 外出接口。客户端：192.168.1.110 / 192.168.2.110

服务端：192.168.1.33 /192.268.2.33

当连接x.x.1.33时，客户端外出接口使用x.x.1.100

当连接x.x.2.33时，客户端外出接口使用x.x.2.100

### UNIX域套接字

1. 特点：
2. 本机传输速度是TCP套接字2倍
3. 可以传递文件描述符
4. 使用路径表示协议族描述符
5. socketpair

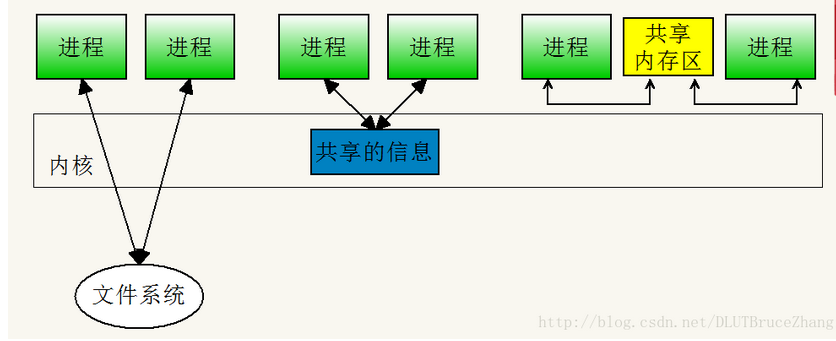
|  |
| --- |
| 1 #include**<**unistd**.**h**>**  2 #include**<**sys**/**socket**.**h**>**  3 #include**<**sys**/**types**.**h**>**  4 #include**<**sys**/**un**.**h**>**  5 #include**<**arpa**/**inet**.**h**>**  6 #include**<**errno**.**h**>**  7  8 #include**<**stdio**.**h**>**  9 #include**<**assert**.**h**>**  10 #include**<**string**.**h**>**  11 int main**(){**  12 int sv**[**2**]** **=** **{**0**};**  13 assert**(**socketpair**(**AF\_UNIX**,**SOCK\_STREAM**,**0**,**sv**)** **!=** **-**1**);**  14 pid\_t pid **=** 0**;**  15 assert**((**pid **=** fork**())!=-**1**);**  16 **if(**pid **>** 0**){**  17 close**(**sv**[**1**]);**  18 int val **=** 0**;**  19 **while(**1**){**  20 **++**val**;**  21 printf**(**"sending val: %d\n"**,**val**);**  22 write**(**sv**[**0**],&**val**,sizeof(**val**));**  23 read**(**sv**[**0**],&**val**,sizeof(**val**));**  24 printf**(**"receive val: %d\n"**,**val**);**  25 sleep**(**1**);**  26 **}**  27 **}**  28 **else{**  29 int val **=** 0**;**  30 close**(**sv**[**0**]);**  31 **while(**1**){**  32 read**(**sv**[**1**],&**val**,sizeof(**val**));**  33 **++**val**;**  34 write**(**sv**[**1**],&**val**,sizeof(**val**));**  35 **}**  36 **}**  37 **return** 0**;**  38 **}** |

1. sendmsg、recvmsg

|  |
| --- |
| struct msghdr**{**  void**\*** msg\_name**;**  socklen\_t msg\_namelen**;**  struct iovec**\*** msg\_iov**;**  size\_t msg\_iovlen**;**  void**\*** msg\_control**;**  size\_t msg\_controllen**;**  int msg\_flags**;**  **};**  struct iovec**{**  void**\*** iov\_base**;**  size\_t iov\_len**;**  **};**  struct cmsghdr**{**  socklen\_t cmsg\_len**;**  int cmsg\_level**;**  int cmsg\_type**;**  /\*char data[]\*/  **};**  **<**fcntl**.**h**>**  CMSG\_FIRSTHDR**(**struct msghdr**\*** mh**);**//返回mh指向的第一个cmsg  CMSG\_NXTHDR**(**struct cmsghdr**\*** cmh**);**//返回下一个cmsg  CMSG\_SPACE**(sizeof(**描述符**));**//根据描述符得到cmsghdr大小  CMSG\_LEN**(sizeof(**描述符**));**//不包括cmsg之间的填充数据  CMSG\_DATA**(**struct cmsghdr**\*** cmsg**);**//指向cmsg种data区 |
| 1 #include**<**unistd**.**h**>**  2 #include**<**sys**/**socket**.**h**>**  3 #include**<**sys**/**types**.**h**>**  4 #include**<**sys**/**un**.**h**>**  5 #include**<**arpa**/**inet**.**h**>**  6 #include**<**errno**.**h**>**  7 #include**<**fcntl**.**h**>**  8  9 #include**<**stdio**.**h**>**  10 #include**<**assert**.**h**>**  11 #include**<**string**.**h**>**  12 void send\_msg**(**int sockfd**,**int fd**){**  13 struct msghdr mh**;**  14 struct iovec iov**;**  15 struct cmsghdr**\*** cmh**;**  16  17 char buf**[**1**]** **=** **{**0**};**  18 iov**.**iov\_base **=** buf**;**  19 iov**.**iov\_len **=** **sizeof(**buf**);**  20  21 char msg\_ctrl**[**CMSG\_SPACE**(sizeof(**fd**))]** **=** **{**0**};**  22 mh**.**msg\_control **=** msg\_ctrl**;**  23 mh**.**msg\_controllen **=** **sizeof(**msg\_ctrl**);**  24 cmh **=** CMSG\_FIRSTHDR**(&**mh**);**  25 cmh**->**cmsg\_len **=** CMSG\_LEN**(sizeof(&**fd**));**  26 cmh**->**cmsg\_level **=** SOL\_SOCKET**;**  27 cmh**->**cmsg\_type **=** SCM\_RIGHTS**;**  28 **\*((**int**\*)**CMSG\_DATA**(**cmh**))** **=** fd**;**  29  30 mh**.**msg\_name **=** **NULL;**  31 mh**.**msg\_namelen **=** 0**;**  32 mh**.**msg\_iov **=** **&**iov**;**  33 mh**.**msg\_iovlen **=** 1**;**  34 mh**.**msg\_flags **=** 0**;**  35 int ret **=** sendmsg**(**sockfd**,&**mh**,**0**);**  36 printf**(**"sendmsg size : %d\n"**,**ret**);**  37 **}**  38 int recv\_msg**(**int sockfd**){**  39 int fd **=** **-**1**;**  40 struct msghdr mh**;**  41 struct cmsghdr**\*** cmh**;**  42 struct iovec iov**;**  43  44 char buf**[**1**]** **=** **{**0**};**  45 iov**.**iov\_base **=** buf**;**  46 iov**.**iov\_len **=** **sizeof(**buf**);**  47  48 char msg\_ctrl**[**CMSG\_SPACE**(sizeof(**fd**))];**  49 mh**.**msg\_control **=** msg\_ctrl**;**  50 mh**.**msg\_controllen **=** **sizeof(**msg\_ctrl**);**  51 cmh **=** CMSG\_FIRSTHDR**(&**mh**);**  52 cmh**->**cmsg\_len **=** CMSG\_LEN**(sizeof(**fd**));**  53 cmh**->**cmsg\_level **=** SOL\_SOCKET**;**  54 cmh**->**cmsg\_type **=** SCM\_RIGHTS**;**  55 **\*((**int**\*)**CMSG\_DATA**(**cmh**))** **=** fd**;**  56  57 mh**.**msg\_name **=** **NULL;**  58 mh**.**msg\_namelen **=** 0**;**  59 mh**.**msg\_iov **=** **&**iov**;**  60 mh**.**msg\_iovlen **=** 1**;**  61 mh**.**msg\_flags **=** 0**;**  62  63 int ret **=** recvmsg**(**sockfd**,&**mh**,**0**);**  64 printf**(**"recv %d\n"**,**ret**);**  65 fd **=** **\*((**int**\*)**CMSG\_DATA**(**cmh**));**  66 assert**(**fd **!=** **-**1**);**  67 **return** fd**;**  68 **}**  69 int main**(){**  70 int sv**[**2**]** **=** **{**0**};**  71 assert**(**socketpair**(**AF\_UNIX**,**SOCK\_STREAM**,**0**,**sv**)** **!=** **-**1**);**  72 pid\_t pid **=** 0**;**  73 assert**((**pid **=** fork**())!=-**1**);**  74 **if(**pid **>** 0**){**  75 close**(**sv**[**1**]);**  76 int fd **=** recv\_msg**(**sv**[**0**]);**  77 char buf**[**1024**]** **=** **{**0**};**  78 read**(**fd**,**buf**,sizeof(**buf**));**  79 fputs**(**buf**,**stdout**);**  80 **}**  81 **else{**  82 close**(**sv**[**0**]);**  83 int fd **=** open**(**"test"**,**O\_RDONLY**);**  84 assert**(**fd**!=-**1**);**  85 send\_msg**(**sv**[**1**],**fd**);**  86 **}**  87 **return** 0**;**  88 **}** |

## 进程间通信

### 进程间通信介绍（一）

1. 互斥与同步
   1. 互斥：不同间进程竞争资源。该资源叫做临界资源；操作该资源的代码代码叫做临界区
   2. 同步：不同进程相互合作完成一项任务
2. 进程间通讯目的
   1. 传输数据
   2. 共享资源（互斥同步）
   3. 事件通知
   4. 控制进程（debug）
3. 发展
   1. 管道
   2. system V
   3. POSIX
4. 分类
   1. 文件（数据传输）
   2. 文件锁（资源共享）
   3. 管道（数据传输）
   4. 信号（事件通知、进程控制）
   5. 消息队列（数据传输）
   6. 共享内存（资源共享）
   7. 信号量（资源共享）
   8. 互斥量（资源共享）
   9. 条件变量（资源共享）
   10. 读写锁（资源共享）
   11. 套接字（tcp、udp、unix域）
5. 共享信息三种方式
6. IPC对象的持续性
   1. 随进程持续
   2. 随内核持续
   3. 随文件系统持续

### 进程间通信介绍（二）

1. 死锁

|  |  |
| --- | --- |
| 构成条件 | * 互斥 * 请求与等待 * 不可剥夺 * 环路等待 |
| 解决方案 | * 资源统一分配：只有请求资源全部可满足时才会分配 * 可抢占 * 资源有序分配：破坏环路等待 |
| 典型算法：  银行家算法  哲学家就餐问题 |  |

1. 信号量和PV原语

|  |
| --- |
| struct semaphore**{**  int value**;**  pointer\_PBC queue**;**  **}**  P**(**s**){**  **--**s**.**value**;**  **if(**s**.**value**<**0**){**  进程入等待队列  **}**  **}**  V**(**s**){**  **++**s**.**value**;**  **if(**s**.**value**<=**0**){**  唤醒进程  **}**  **}** |
| 总结：  P、V操作用于不同进程时，同步；  P、V用于一个进程，互斥 |

1. 实际应用

|  |  |
| --- | --- |
| 公交司机和售票员 | 解法1：  司机进程：  S1**(**0**)**  **while(**1**){**  P**(**S1**);**  开车**;**  停车**;**  V**(**S2**);**  **}**  售票员进程：  S2**(**0**)**  **while(**1**){**  关门**;**  V**(**S1**);**  售票**;**  P**(**S2**);**  开门**;**  **}**  解法2：  司机进程：  S1**(**0**);**  **while(**1**){**  停车  V**(**S2**);**  P**(**S1**);**  开车  **}**  售票员进程：  S2**(**0**);**  **while(**1**){**  P**(**S2**)**  开门**;**  关门；  售票；  V**(**S1**);**  **}** |
| 民航售票 | 民航售票：  余票 **=** x  S**(**1**);**  P**(**S**);**  **if(**x**>**0**)**  **--**x**;**  V**(**S**);**  其中x称为时临界资源；这块代码称为临界区 |
| 汽车租赁：  总共有2辆车，四位顾客 | S**(**2**);**  P**(**S**);**  租车**;**  还车**;**  V**(**S**);** |