# 网络编程教学案例库使用手册

## 所在函数库为allnet.h

1. TCP客户端链接服务器函数（包含服务器地址解析、套接字创建、服务器连接等）；
   1. int client\_connect(int argc, char \*servaddr, char \*port);

该函数返回值是一个已经创建好的套接字，3个参数的含义，

argc：命令行总的参数个数。

servaddr：要连接的目标ip地址。

port：要连接的目标端口号。

1. UDP客户端连接服务器函数（包含服务器地址解析、套接字创建、服务器连接等）；

2.1 int udpclient\_connect(int argc, char \*servaddr)；

该函数返回值是一个已经创建好的套接字，2个参数的含义，

argc：命令行总的参数个数。

servaddr：要连接的目标ip地址。

1. TCP服务器准备函数（包含服务器地址发现、套接字创建、地址绑定、服务器监听等）；

3.1 int tcp\_pre(const char\* servaddr, const char\* port)；

该函数返回值是一个已经创建好的套接字，2个参数的含义，

servaddr：要连接的目标ip地址。

port：要连接的目标端口号。

1. UDP服务器准备函数（包含服务器地址发现、套接字创建、地址绑定、服务器监听等）；
   1. int udp\_pre(const char\* servaddr, const char\* port)；

该函数返回值是一个已经创建好的套接字，2个参数的含义，

servaddr：要连接的目标ip地址。

port：要连接的目标端口号。

1. TCP并发服务器模型（包含多进程服务器、多线程服务器等）；

5.1 子进程构造函数

ssize\_t process\_create()；

该函数的返回值是一个进程号

5.2 serv\_accept(int listenfd, int len, struct sockaddr\_in \*cliaddr)；

该函数指定服务端去接收客户端连接，返回值是客户端套接字的标识。

各个参数意义如下，

listenfd:执行过listen系统调用的监听套接字

len：socket的地址长度

cliaddr：用来获取被接收连接的远端套接字地址

5.3 void resource\_getback(int signo)；

该函数是用于处理僵死进程，尽可能回收资源。1个参数的意义如下，

signo:该参数用于区分信号，必须填写。

1. I/O复用技术TCP服务器（包含select、poll、epoll等）模型

本套服务器模型包括三个，分别是select、poll、epoll模型，通过构造了一个结构体将一些初始数据包含在结构体中。结构体定义如下：

typedef struct

{

fd\_set allset;

fd\_set rset;

int clients[FD\_SETSIZE]; /\*数组用于装载各事件描述符\*/

int maxfd; /\*最大描述符\*/

int maxi;

struct pollfd pclients[1024]; /\*数组用于记录发生的事件\*/

struct epoll\_event events[1024]; /\*数组用于记录发生的事件\*/

int epollfd; /\*epoll句柄的返回值\*/

int nready; /\*创建套接字返回值\*/

int listenfd; /\*接收监听套接字返回值\*/

}select\_s,poll\_p,epoll\_e;

6.1 void select\_pre(select\_s \*s, int listenfd)；

该函数的功能是将结构体中的数据初始化，包括对描述符置0，监听套接字设置到描述符集合，以及最大描述符的初始化。

参数意义：

s:结构体指针

listenfd:执行过listen系统调用的监听套接字

下面是函数体内部

s->listenfd = listenfd;

FD\_ZERO(&s->allset);

FD\_ZERO(&s->rset);

FD\_SET(listenfd, &s->allset);

int i;

for(i = 0; i < FD\_SETSIZE; i++){

s->clients[i] = -1;

}

s->maxi = -1;

s->maxfd = listenfd;

6.2 void select\_do\_wait(select\_s \*s)；

该函数功能是监听用户感兴趣的文件描述符上的可读、可写或异常事件。返回值是就绪描述符的个数。

参数定义，

s:结构体指针

函数体内部如下，

void select\_do\_wait(select\_s \*s){

s->rset = s->allset;

do{

s->nready = select(s->maxfd + 1, &s->rset, NULL, NULL, NULL);

}while(s->nready == -1 && errno == EINTR);

}

6.3 void select\_handle\_accept(select\_s \*s, struct sockaddr\_in \*cliaddr, socklen\_t \*clilen);

该函数功能是测试fdset的位fd是否已经设置为监听套接字，如果是，则接收客户端连接，并且用户文件描述符减1.

参数定义，

s:结构体指针

cliaddr：用来获取被接收连接的远端套接字地址

clilen：socket地址长度

函数体内部如下：

void select\_handle\_accept(select\_s \*s, struct sockaddr\_in \*cliaddr, socklen\_t \*clilen)

{

if(FD\_ISSET(s->listenfd, &s->rset))

{

int peerfd = accept(s->listenfd, (struct sockaddr\*)&cliaddr, clilen);

if(peerfd == -1)

{

printf("accept wrong\n");

exit(1);

}

select\_add\_fd(s, peerfd);

}

s->nready--;

}

6.4 void select\_add\_fd(select\_s \*s, int fd)；

该函数功能是向数组中装载接收了客户端连接的描述符（执行了accept后的返回值进行装载添加）

参数定义，

s:结构体指针

fd：接收了客户端连接的描述符

函数体内部如下：

void select\_add\_fd(select\_s \*s, int fd)

{

int i;

for(i = 0; i < FD\_SETSIZE; i++)

{

if(s->clients[i] == -1)

{

s->clients[i] = fd;

if(i > s->maxi)

s->maxi = i;

break;

}

}

if(i == FD\_SETSIZE)

{

printf("too many clients\n");

close(fd);

exit(1);//!

}

FD\_SET(fd, &s->allset);

if(fd > s->maxfd)

s->maxfd = fd;

}

6.5 void select\_del\_fd(select\_s \*s, int i)；

该函数的功能是初始化装载描述符的数组

参数定义，

s:结构体指针

i：数组下标

函数体内部如下：

void select\_del\_fd(select\_s \*s, int i)

{

assert(i >= 0 && i < FD\_SETSIZE); //!

int fd = s->clients[i];

s->clients[i] = -1;

FD\_CLR(fd, &s->allset);

close(fd);

}

6.5 void select\_close(select\_s \*s)

该函数功能是关闭已创建的套接字

参数定义，

s:结构体指针

函数体内部如下：

void select\_close(select\_s \*s)

{

close(s->listenfd);

}

6.6 void poll\_pre(poll\_p \*p, int listenfd)

该函数的功能是将结构体中的数据初始化，包括设置监听套接字、初始化数组、设置成员描述符为监听套接字、注册POLLIN、初始化被监听集合大小。

参数意义：

s:结构体指针

listenfd:执行过listen系统调用的监听套接字

函数体内部如下：

void poll\_pre(poll\_p \*p, int listenfd){

p->listenfd = listenfd;

int i;

for(i = 0; i < 1024; i++){

p->pclients[i].fd = -1;

}

p->pclients[0].fd = listenfd;

p->pclients[0].events = POLLIN;

p->maxi = 0;

p->nready = 0;

}

6.7 void poll\_do\_wait(poll\_p \*p)；

该函数功能是轮训一定数量的文件描述符，测试文件描述符集是否有就绪者。

参数意义：

p:结构体指针

函数体内部如下：

void poll\_do\_wait(poll\_p \*p){

int nready;

do{

nready = poll(p->pclients, p->maxi + 1, -1);

}while(nready == -1 && errno == EINTR);

if(nready == -1)

{

printf("poll wait wrong\n");

exit(1);

}

p->nready = nready;

}

6.8 void poll\_handle\_accept(select\_s \*s, struct sockaddr\_in \*cliaddr, socklen\_t \*clilen);

该函数功能是如果数字中装载的描述符为POLLIN事件则接收客户端连接并且就绪描述符个数减1

参数定义，

p:结构体指针

cliaddr：用来获取被接收连接的远端套接字地址

clilen：socket地址长度

函数体内部如下：

void poll\_handle\_accept(poll\_p \*p, struct sockaddr\_in \*cliaddr, socklen\_t \*clilen)

{

if(p->pclients[0].revents & POLLIN){

int peerfd = accept(p->listenfd, (struct sockaddr\*)&cliaddr, clilen);

if(peerfd == -1)

{

printf("accept wrong\n");

exit(1);

}

poll\_add\_fd(p, peerfd);

--p->nready;

}

}

6.9 void poll\_add\_fd(poll\_p \*p, int fd)；

该函数功能是向数组中装载接收了客户端连接的描述符（执行了accept后的返回值进行装载添加），对注册事件设置POLLIN

参数定义，

p:结构体指针

fd：接收了客户端连接的描述符

函数体内部如下：

void poll\_add\_fd(poll\_p \*p, int fd){

int i;

for(i = 0; i < 1024; i++){

if(p->pclients[i].fd == -1){

p->pclients[i].fd = fd;

p->pclients[i].events= POLLIN;

if(i > p->maxi)

p->maxi = i;

break;

}

}

if(i == 1024){

fprintf(stderr, "too many pclients\n");

close(fd);

exit(1);

}

}

6.10 void poll\_del\_fd(poll\_p \*p, int i)；

该函数的功能是初始化装载描述符的数组

参数定义，

p:结构体指针

i：数组下标

函数体内部如下：

void poll\_del\_fd(poll\_p \*p, int i){

assert(i >= 1 && i < 1024);//

close(p->pclients[i].fd);

p->pclients[i].fd = -1;

}

6.11 void poll\_close(poll\_p \*p)

该函数功能是关闭已创建的套接字

参数定义，

p:结构体指针

函数体内部如下：

void poll\_close(poll\_p \*p)

{

close(p->listenfd);

}

6.12 void epoll\_pre(epoll\_e \*e, int listenfd)

该函数的功能是将结构体中的数据初始化，包括生成epoll句柄，监听套接字初始化、往内核注册表中注册被监听的事件。

参数意义：

e:结构体指针

listenfd:执行过listen系统调用的监听套接字

函数体内部如下：

void epoll\_pre(epoll\_e \*e, int listenfd)

{

if((e->epollfd = epoll\_create(1024)) == -1)

{

printf("epoll create wrong\n");

exit(1);

}

e->listenfd = listenfd;

epoll\_add\_fd(e, listenfd);

e->nready = 0;

}

6.13 void epoll\_do\_wait(epoll\_e \*e)

该函数的功能是等待一组文件描述符上的事件。如果检查到事件，就将所有就绪事件从内核事件表中复制到events数组中。

参数意义：

e:结构体指针

函数体内部如下：

void epoll\_do\_wait(epoll\_e \*e)

{

int nready;

do{

nready = epoll\_wait(e->epollfd, e->events, 10242, -1);

}while(nready == -1 && errno == EINTR);

if(nready == -1)

{

printf("epoll wait wrong\n");

exit(1);

}

e->nready = nready;

}

void epoll\_handle\_accept(epoll\_e \*e)

该函数功能是指定服务端去接收客户端连接，返回值是客户端套接字的标识，并将该事件注册到内核事件表。

参数意义：

e:结构体指针

函数体内部如下：

void epoll\_handle\_accept(epoll\_e \*e, struct sockaddr\_in \*cliaddr, socklen\_t \*clilen)

{

int peerfd = accept(e->listenfd, (struct sockaddr\*)&cliaddr, clilen);

if(peerfd == -1)

{

printf("accept wrong\n");

exit(1);

}

epoll\_add\_fd(e, peerfd);

}

6.14 void epoll\_add\_fd(epoll\_e \*e, int fd)

该函数功能是向内核的事件表中添加事件。

参数定义，

e:结构体指针

fd：接收了客户端连接的描述符

函数体内部如下：

void epoll\_add\_fd(epoll\_e \*e, int fd)

{

struct epoll\_event ev;

ev.data.fd = fd;

ev.events = EPOLLIN;

if(epoll\_ctl(e->epollfd, EPOLL\_CTL\_ADD, fd, &ev) == -1)

{

printf("ctl wrong\n");

exit(1);

}

}

6.15 void epoll\_del\_fd(epoll\_e \*e, int fd)

该函数功能是将指定事件从内核事件表中删除。

参数定义，

e:结构体指针

fd：接收了客户端连接的描述符

函数体内部如下：

void epoll\_del\_fd(epoll\_e \*e, int fd)

{

struct epoll\_event ev;

ev.data.fd = fd;

if(epoll\_ctl(e->epollfd, EPOLL\_CTL\_DEL, fd, &ev) == -1)

{

printf("del wrong\n");

exit(1);

}

}

6.16 void epoll\_close(epoll\_e \*e)

该函数功能是删除epoll句柄，关闭监听套接字。

参数定义，

e:结构体指针

函数体内部如下：

void epoll\_close(epoll\_e \*e)

{

close(e->epollfd);

close(e->listenfd);

}

1. 进程池服务器模型

7.1 void poll\_resource\_back(int nchildren, pid\_t \*pids)

该函数的功能是回收进程池中的进程，也就是回收资源。

参数定义，

nchildren:进程池中的进程数

pids：进程号

函数体内部如下：

void poll\_resource\_back(int nchildren, pid\_t \*pids)

{

int i;

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

kill(pids[i], SIGTERM);

while (wait(NULL) > 0) /\* wait for all children \*/

;

if (errno != ECHILD)

printf("wait error");

}

1. 线程池服务器模型

8.1 void pthread\_poll\_create(Thread \*tptr, int nthreads, void(\*thread\_main)(void \*))

该函数功能是根据用户指定数目派生出线程池。

参数定义，

tptr:结构体指针

nchildren:进程池中的进程数

thread main:线程处理工作的地址入口

函数体内部如下：

void pthread\_poll\_create(Thread \*tptr, int nthreads, void(\*thread\_main)(void \*))

{

int i;

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

{

pthread\_create(&tptr[i].thread\_tid, NULL, thread\_main, (void \*) i);

}

}

8.2 int pthread\_lock\_accept(pthread\_mutex\_t \*mutex, int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen)

该函数主要是用在线程池内线程各accept，线程工作之前要调用accept，前后必须加锁加以保护。函数的返回值是接收客户端连接的客户端套接字标识

参数定义，

mutex：线程锁

sockfd：监听套接字

addr：用来获取被接收连接的远端套接字地址

addrlen：socket套接字长度

函数体内部如下：

int pthread\_lock\_accept(pthread\_mutex\_t \*mutex, int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen)

{

int connfd;

pthread\_mutex\_lock(&mutex);

connfd = accept(sockfd, addr, addrlen);

pthread\_mutex\_unlock(&mutex);

return connfd;

}

8.3 int lock(pthread\_mutex\_t \*chifd\_mutex)

该函数主要用于线程池服务模型主线程统一accept，线程池中的线程获得互斥锁。主线程接收一格连接后调用pthread\_cond\_signal向条件变量发送信号，以唤醒睡眠在其上的线程。该函数的返回值是一个整数，返回0表示成功，其它任何值表示失败。

参数定义：

chifd\_mutex：线程锁指针

函数体内部如下：

int lock(pthread\_mutex\_t \*clifd\_mutex)

{

return pthread\_mutex\_lock(clifd\_mutex);

}

8.4 int unlock(pthread\_mutex\_t \*clifd\_mutex)

该函数主要用于线程池服务模型主线程统一accept，释放线程锁。该函数的返回值是一个整数，返回0表示成功，其它任何值表示失败。

参数定义：

chifd\_mutex：线程锁指针

函数体内部如下：

int unlock(pthread\_mutex\_t \*clifd\_mutex)

{

return pthread\_mutex\_unlock(clifd\_mutex);

}

8.5 void s\_signal(int connfd)

该函数主要用于线程池服务模型主线程统一accept，线程池中的线程获得互斥锁之后测试iput和iget，若两者相等则无事可做，调用pthread\_cond\_signal唤醒睡眠的线程。

参数定义，

connfd：接收客户端连接的客户端套接字标识（accept函数的返回值）

函数体内部如下：

void s\_signal(int connfd)

{

clifd[iput] = connfd;

if (++iput == MAXNCLI)

iput = 0;

if (iput == iget)

{

printf("iput = iget = %d", iput);

exit(0);

}

pthread\_cond\_signal(&clifd\_cond);

}

8.6 int w\_wait(int connfd)

该函主要功能是用于阻塞当前线程，等待有线程调用pthread\_cond\_signal唤醒。

函数返回值是接收客户端连接的客户端套接字标识（accept函数的返回值）。

参数定义，

connfd：接收客户端连接的客户端套接字标识（accept函数的返回值）。

函数体内部如下：

int w\_wait(int connfd)

{

while (iget == iput)

pthread\_cond\_wait(&clifd\_cond, &clifd\_mutex);

connfd = clifd[iget]; /\* connected socket to service \*/

if (++iget == MAXNCLI)

iget = 0;

return connfd;

}

1. 其它工作函数

9.1 void server\_work(int connfd)

该函数的功能是服务器接收客户发来的数据，并回写数据。主要是用了搭建简单的回射服务器。

参数定义，

connfd: 客户端套接字的标识(accept的返回值)

函数体内部如下：

void server\_work(int connfd)

{

int ntowrite;

ssize\_t nread;

char line[MAXLINE], result[MAXN];

while(1)

{

if ((nread = Readline(connfd, line, MAXLINE)) == 0)

return; /\* connection closed by other end \*/

/\* 4line from client specifies #bytes to write back \*/

ntowrite = atol(line);

if ((ntowrite <= 0) || (ntowrite > MAXN))

{

printf("client request for %d bytes\n", ntowrite);

exit(0);

}

printf("server get data : %s", line);

Writen(connfd, result, ntowrite);

}

}

9.2 ssize\_t readline(int fd, void \*vptr, size\_t maxlen)

该函数功能是接收客户端发来的数据，函数会把参数fd所指的文件传送到vptr所指内存中。

参数定义

fd：客户端套接字的标识(accept的返回值)

vptr：缓冲区指针

maxlen：能接受数据的最大长度

函数体内部如下：

ssize\_t Readline(int fd, void \*ptr, size\_t maxlen)

{

ssize\_t n;

if ((n = readline(fd, ptr, maxlen)) == -1)

printf("readline error\n");

return(n);

}

9.3 void Writen(int fd, void \*ptr, size\_t nbytes)

该函数功能是向客户端回写数据，该函数会把参数ptr所指内存写入nbytes个字节到参数fd

fd：客户端套接字的标识(accept的返回值)

ptr：缓冲区指针

nbytes：回写数据大小

函数体内部如下：

void Writen(int fd, void \*ptr, size\_t nbytes)

{

if (writen(fd, ptr, nbytes) != nbytes)

printf("writen error");

}