常见函数

```
inet_aton("127.0.0.1",&addr.sin_addr)    //字符串转整形
s = inet_ntoa(addr.sin_addr)             //整形转字符串
```

```
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
//h表示host，n表示net，s表示short，l表示long
```

```
struct sockaddr_in addr;
socklen_t len;
getsockname(sockfd,(struct sockaddr *)&addr,&len);  //本地套接字信息
printf("addr: %s, %d\n",inet_ntoa(addr.sin_addr),ntohs(addr.sin_port));
//getpeername相似，获取对方套接字信息
```

```
struct hontent *he = gethostbyname(address);
if(he == NULL)  return -1;
*inaddr = *(struct in_addr*)he->h_addr_list[0];
//可有多个
```

```
int on=1;
setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR, &on,sizeof(int));
//地址重用
```

```
int on = 1;
setsockopt(sockfd,SOL_SOCKST,SO_KEEPALIVE,&on,sizeof(int));
//保持长连接
```

```
int nIO = 1;
ioctl(fd,FIONBIO,&nIO);
//设置套接字为非阻塞方式
```

```
struct sigaction act;
act.sa_handler = 函数名;   //SIG_DEL默认，SIG_IGN忽略
sigemptyset(&act.sa_mask);
act.sa_flags = 0;
sigaction(SIGALRM,&act,NULL);
```

TCP循环（超时处理）

服务器

```
int timeout_flag;
void sigalrm_handler(int sig)
{
    //超时处理
}

int main(int argc, char *argv[])
{
    struct sigaction act;
    act.sa_handler = sigalrm_handler;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    sigaction(SIGALRM,&act,NULL);

    struct sockaddr_in s_addr,c_addr;
    int s_fd = socket(AF_INET,SOCK_STREAM,0);   //创建套接字
    if(s_fd == -1)
    {
        cout << "socket fail" << endl;
        return 1;
    }
    memset(s_addr,0,sizeof(s_addr));
    s_addr.sin_family = AF_INET;
    s_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    s_addr.sin_port = htons(PORT);
    if(bind(s_fd,(struct sockaddr *)&s_addr,sizeof(struct sockaddr)) == -1) //绑定地址
    {
        cout << "bind socket fail" << endl;
        return 1;
    }
    if(listen(s_fd,BACKLOG) == -1)                //监听端口
    {
        cout << "listen soctet fail" << endl;
        return 1;
    }
    socklen_t c_len = sizeof(c_addr);
    int c_fd = accept(s_fd,(struct sockaddr *)&c_addr,&c_len);  //接受客户端连接
    if(c_fd == -1)
    {
        cout << "accept soctet fail" << endl;
        return 1;
    }
    char buf[1005];
    while(1)
    {
        timeout_flag = 0;
        alarm(20);
        int len = read(c_fd,buf,1000);
        alarm(0);
        if(len < 0)
        {
            if(timeout_flag)
            {
                //超时处理...
            }
            else
            {
                //出错处理...
            }
        }
        else
        {
            //正常处理...
        }
    }
    close(c_fd);
    close(f_fd);
    return 0;
}
```

客户端

```
int main(int argc, char *argv[])
{
    struct sockaddr_in addr;
    memset(addr,0,sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    inet_aton("127.0.0.1",&addr.sin_addr);
    int c_fd = socket(AF_INET,SOCK_STREAM,0);   //建立套接字
    if(c_fd == -1)
    {
        cout << "socket fail" << endl;
        return 0;
    }
    if(connect(c_fd,(struct sockaddr *)&s_addr,sizeof(sockaddr)) == -1) //连接服务器
    {
        cout << "connect fail" << endl;
        return 0;
    }
    char buf[1005];
    int len = read(c_fd,buf,sizeof(buf));       //接收
    write(c_fd,buf,sizeof(buf));                //发送
    char buf[1005];
    close(c_fd);
    return 0;
}
```

TCP并发
（多进程）

```
int main(int argc, char const *argv[])
{
    //套接字与地址初始化...
    while(1)
    {
        socklen_t len;
        c_fd = accept(s_fd,(struct sockaddr *)&c_addr,&c_len);
        if(c_fd == -1)
        {
            cout << "accept soctet fail" << endl;
            return 1;
        }
        pid_t pid = fork();
        if(pid == -1)
        {
            cout << "fork error" << endl;
            return 1;
        }
        else if(pid == 0)   //子进程
        {
            close(s_fd);
            while(1)
            {
                //数据收发...
            }
            close(c_fd);
        }
        else close(c_fd);   //父进程
    }
    close(s_fd);
    return 0;
}
```

TCP并发（预创建子进程）

```
int pids[CLDNUM+1];

void theEnd(int n)
{
    for(int i = 1;i <= n;i++)
    {
        if(pids[i] > 0) kill(pids[i],SIGTERM);
    }
    while(wait(NULL) > 0);
}

int pids[NUM+1];

int main(int argc, char const *argv[])
{
    int listenfd,connfd;
    if((listenfd = socket(AF_INET,SOCK_STREAM,0)) < 0)
    {
        cout << "socket error" << endl;
        return 1;
    }
    struct sockaddr_in addr;
    memset(s_addr,0,sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    addr.sin_port = htons(PORT);
    int on = 1;
    setsockopt(listenfd,SOL_SOCKET,SO_REUSEADDR,&on);    //地址重用
    if(bind(listenfd,(struct sockaddr *)&addr,sizeof(struct sockaddr)) == -1)
    {
        cout << "bind socket fail" << endl;
        return 1;
    }
    if(listen(listenfd,BACKLOG) == -1)                   //监听端口
    {
        cout << "listen soctet fail" << endl;
        return 1;
    }
    int nErr = 0;
    for(int i = 1;i <= NUM;i++)
    {
        if((pids[i] = fork()) < 0)
        {
            nErr = i;
            break;
        }
        else if(pids[i] == 0)
        {
            while(1)
            {
                connfd = accept(listenfd,NULL,NULL);
                //read、write、....
            }
        }
    }
    if(nErr != NUM)
    {
        theEnd(nErr-1);
        return 1;
    }
    char s[100];
    while(cin >> s)
    {
        if(strcmp(s,"end") == 0)
        {
            theEnd(nErr);
            return 0;
        }
    }
    return 0;
}
```

TCP非阻塞

```
int main(int argc, char const *argv[])
{
    //套接字与地址初始化...
    int on = 1;
    ioctl(sockfd,FINNBIO,&on);
    while(1)
    {
        int new_fd = accept(sockfd,NULL,NULL);
        if(new_fd == -1)
        {
            cout << "accept error" << endl;
            sleep(1);
        }
        else
        {
            //读写操作...
        }
        close(new_fd);
    }
    return 0;
}
```

多路复用

```
int main(int argc, char const *argv[])
{
    int sockfd[NUM+1];
    struct sockaddr_in addr[NUM+1];
    fd_set rfds;
    char buf[1005];
    for(int i = 1;i <= NUM;i++)
    {
        sockfd[i] = socket(AF_INET,SOCK_STREAM,0);
        if(sockfd[i] < 0)
        {
            cout << "socket error" << endl;
            return 1;
        }
    }
    //填充NUM个地址，建立NUM个连接...
    int nOK[NUM+1] = {0};
    while(1)
    {
        FD_ZERO(rfds);
        int maxx = 0;
        for(int i = 1;i <= NUM;i++)
        {
            if(nOK[i] == 0)
            {
                FD_SET(sockfd[i],&rfds);
                maxx = max(maxx,sockfd[i]);
            }
        }
        int n = select(maxx+1,&rfds,NULL,NULL,NULL);
        if(n < 0)
        {
            cout << "select error" << endl;
            return 1;
        }
        for(int i = 1;i <= NUM;i++)
        {
            if(FD_ISSET(sockfd[i],&rfds))
            {
                //该套接字操作...
            }
        }
    }
    for(int i = 1;i <= num;i++) close(sockfd[i]);
    return 0;
}
```

UDP广播
（超时处理）

```
int timeout_flag;
void sigalrm_handler(int signo)
{
    timeout_flag = 1;
}
int main(int argc, char const *argv[])
{
    struct sigaction act;
    act.sa_handler = sigalrm_handler;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    sigaction(SIGALRM,&act,NULL);

    int sockfd = socket(AF_INET,SOCK_DGRAM,0);
    if(sockfd == -1)
    {
        cout << "create socket error" << endl;
        return 1;
    }
    sockaddr_in addr;
    memset(addr,0,sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(PORT);
    addr.sin_addr.s_addr = htonl(ADDR_X);   //广播地址
    char buf[1005];
    while(1)
    {
        memset(buf,0,sizeof(buf));
        sendto(sockfd,buf,strlen(buf),0,(struct sockaddr *)&addr,sizeof(addr));    //广播
        while(1)
        {
            sockaddr_in c_addr;
            int len;
            timeout_flag = 0;
            alarm(5);
            int n = recvfrom(sockfd,buf,1000,0,(struct sockaddr*)&c_addr,&len);
            alarm(0);
            if(n < 0)
            {
                if(timeout_flag == 1)
                {
                    //超时处理...
                }
                else
                {
                    //出错处理...
                }
            }
            else
            {
                //正常处理...
            }
        }
    }
    close(sockfd);
    return 0;
}
```

UDP信号驱动

```c
int sockfd;

void sigio_handler(int signo)
{
    struct sockaddr_in addr;
    socklen_t len;
    char buf[1024];
    int n = recvfrom(sockfd,buf,1000,0,(struct sockaddr *)&addr,&len);
    if(n > 0)
    {
        //数据处理...
    }
    else
    {
        //错误处理...
    }
}

int main(int argc,char **argv)
{
    //UDP套接字初始化...
    signal(SIGIO,sigio_handler);
    pid_t pid = getpid();
    ioctl(sockfd,FIOSETOWN,&pid);    //设置属主
    int on = 1;
    ioctl(sockfd,FIOASYNC,&on);      //启动信号驱动模式
    while(1)    sleep(1);
    close(sockfd);
    return 0;
}
```

原始套接字
ping

```
int timeout_flag;
void sigalrm_hander(int sig)
{
    timeout_flag = 1;
}
int main(int argc, char const *argv[])
{
    struct sigaction act;
    act.sa_handler = sigalrm_handler;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    sigaction(SIGALRM,&act,NULL);

    int sockfd = socket(AF_INET,SOCK_RAW,IPPROTO_ICMP); //创建套接字
    if(sockfd == -1)
    {
        cout << "socket fail" << endl;
        return 1;
    }
    struct sockaddr_in addr;
    memset(addr,0,sizeof(addr));
    inet_aton("xx.xx.xx.xx",&addr.sin_addr);
    int cnt = 0;
    unsigned long buf[64];
    while(cnt++ < 3)
    {
        timeval tv;
        gettimeofday(&tv,NULL);         //获取当前时间
        buf[2] = htonl(tv.sec);
        buf[3] = htonl(tv.usec);
        FillIcmpHdr((char *)buf,8);     //填充数据包
        int len = sizeof(addr);
        sendto(sockfd,buf,128,0,(struct sockaddr *)&addr,len);
        timeout_flag = 0;
        alarm(5);
        int n = recvfrom(sockfd,buf,128,0,(struct sockaddr *)&addr,&len);
        alarm(0);
        if(n < 0)
        {
            if(timeout_flag == 1)
            {
                //超时处理...
            }
            else
            {
                //错误处理...
            }
        }
        else    recv_process(buf);  //计算校验和，响应时间
    }
    close(sockfd);
    return 0;
}
```

管道

```cpp
int main(int argc, char const *argv[])
{
    int fds1[2],fds2[2];
    if(pipe(fd1s) < 0)
    {
        cout << "pipe error" << endl;
        return 1;
    }
    if(pipe(fd2s) < 0)
    {
        cout << "pipe error" << endl;
        return 1;
    }
    pid_t pid = fork();
    if(pid < 0)
    {
        cout << "fork error" << endl;
        return 1;
    }
    if(pid > 0)     //父进程
    {
        close(fds1[0]);     //关fds1的读
        close(fds2[1]);     //关fds2的写
        write(fds2[1],buf,100);
        read(fd1[0],buf,100);
        close(fds1[1]);
        close(fds2[0]);
    }
    else            //子进程
    {
        close(fds1[1]);     //关fds1的写
        close(fds2[0]);     //关fds2的读
        write(fds1[1],buf,100);
        read(fd2[0],buf,100);
        close(fds1[0]);
        close(fds2[1]);
    }
    return 0;
}
```

命名管道

```cpp
#define FIFO_SERVER "tmp/fifo"

int main(int argc, char const *argv[])
{
    if(mkfifo(FIFO_SERVER,O_CREAT|O_EXCL) < 0 && errno != EEXIST)   //创建
    {
        cout << "create error" << endl;
        return 1;
    }
    int fd;
    fd = open(FIFO_SERVER,O_WRONLY|O_NONBLOCK); //只写、非阻塞
    write(fd,buf,sizeof(buf));
    //fd = open(FIFO_SERVER,O_RDONLY|O_NONBLOCK);   另一端只读、非阻塞
    read(fd,buf,1024);
    close(fd);
    return 0;
}
```

共享内存

```cpp
int main()
{
    key_t key = ftok("/tmp/shml",0);    //创建关键字
    if(key == -1)
    {
        cout << "key error" << endl;
        return 1;
    }
    int shmid = shmget(key,1024,0777);  //创建共享内存段
    if(shmid == -1)
    {
        cout << "create share memory error" << endl;
        return 1;
    }
    char *p_addr,*c_addr;
    char buf[1024];
    if(fork())  //父进程
    {
        p_addr = shmat(shmid,0,0);      //映射地址段
        memset(p_addr,0,1024);
        strcpy(p_addr,buf,sizeof(buf)); //父进程写
    }
    else        //子进程
    {
        c_addr = shmat(shmid,0,0);      //子进程读
        cout << c_addr << endl;
        shmdt(c_addr);                  //解除映射
        exit(0);
    }
    shmdt(p_addr);
    return 0;
}
```

守护进程

```
int init_daemon(void)        //初始化守护进程
{
    int pid = getpid();
    if(pid == fork())    exit(0);                    //结束父进程
    else if(pid < 0)     exit(1);          //fork失败
    setsid();                                        //脱离控制终端、登录会话和进程组
    if(pid == fork())    exit(0);                    //结束第一子进程
    else if(pid < 0)     exit(1);          //fork失败
    for(int i = 0;i < NOFILE;i++)    close(i);    //关闭打开的文件描述符
    chdir("/tmp");                                   //改变当前工作目录
    umask(0);                                        //重设文件权限掩码
    signal(SIGCHLD,SIG_IGN);                         //屏蔽SIGCHLD信号
    return 0;
}
```

带外数据

```
pid_t pid = getpid();
ioctl(sockfd,FIOSETOWN,&pid);
//设置属主，收到带外数据时发送SIGURG给该属主

send(sockfd,”a”,1,MSG_OOB);
//发送带外数据
recv(sockfd,buf,1,MSG_OOB);
//未设置SO_OOBINLINE选项接收带外数据（缺省）
read(sockfd,buf,1);
//设置了SO_OOBINLINE选项接收带外数据

int flag;
ioctl(sockfd,SIOCATMARK,&flag);
//检测带外标记，有flag为1，无flag为0
```