

A Quick Guide to Networking Software

by J. Sanguino

5<sup>th</sup> Edition: February 2020

# Mission Briefing

Welcome to this quick guide in networking programming.

You will be given a username and password to access any of the RC lab computers. They are connected to Internet and running Linux.

Your mission, should you decide to accept it, is to complete the tasks that will be presented as you move along the guide. They involve the development of programs that communicate through the Internet.

The tools that you will be using are the basis for the development of network applications over the Internet (web browsers and servers, email, peer-to-peer, remote logins, file transfers ...).

The kind of network applications you will be able to develop, on your own, at the end of this guide, will only be bounded by your imagination.

As always, should you or any team member be caught in thrall of network programming, the author would disavow any knowledge of your actions.



Login: alunos

Password: alunos

Good! Move on!

2<sup>nd</sup> Task: Now that you have a name, get the IP address.

15 minutes.

# getaddrinfo

```
// test.c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <string.h>

int main(void)
{
    struct addrinfo hints,*res,*p;
    int errcode;
    char buffer[INET_ADDRSTRLEN];
    struct in_addr *addr;

    memset(&hints,0,sizeof hints);
    hints.ai_family=AF_INET;//IPv4
    hints.ai_socktype=SOCK_DGRAM;
    hints.ai_flags=AI_CANONNAME;

    if((errcode=getaddrinfo("tejo.tecnico.ulisboa.pt",NULL,&hints,&res))!=0)
        fprintf(stderr,"error: getaddrinfo: %s\n",gai_strerror(errcode));
    else{
        printf("canonical hostname: %s\n",res->ai_canonname);
        for(p=res;p!=NULL;p=p->ai_next){
            struct sockaddr_in *ip = (struct sockaddr_in *)p->ai_addr;
            addr = &(ip->sin_addr);
            printf("internet address: %s (%#08lx)\n",
                inet_ntop(p->ai_family, addr, buffer, sizeof(buffer)),
                (long unsigned int)ntohl(addr->s_addr));
        }
    }
    freeaddrinfo(res);
}
exit(0);
}
```

More?

```
$ $ man getaddrinfo inet_ntop memset ntohl 7 ip$
```

```
#include <arpa/inet.h>

const char *inet_ntop(int af,
                      const void *src,char *dst,
                      socklen_t size);
```

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

```
int getaddrinfo(const char *node, const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);
void freeaddrinfo(struct addrinfo *res);
const char *gai_strerror(int errcode);
```

```
struct addrinfo {           // (item in a linked list)
    int             ai_flags;   // additional options
    int             ai_family;  // address family
    int             ai_socktype; // socket type
    int             ai_protocol; // protocol
    socklen_t       ai_addrlen; // address length (bytes)
    struct sockaddr *ai_addr;  // socket address
    char            *ai_canonname; // canonical hostname
    struct addrinfo *ai_next;  // next item
};
```

```
#include <string.h>
void *memset(void *s,int c,size_t n);
```

```
struct sockaddr_in {
    sa_family_t     sin_family; // address family: AF_INET
    u_int16_t       sin_port;   // port in (16 bits)
    struct in_addr sin_addr;  // internet address
};
```

0xC1==193  
0x88==136  
0x8A==138  
0x8E==142

network byte order

```
struct in_addr{
    uint32_t        s_addr; // 32 bits
};
```

```
#include <arpa/inet.h>
uint32_t htonl(uint32_t netlong);
(network to host long)
```

Long (32 bits) 0x76543210

Little endian system Network byte order

ADDR	0x10	ADDR	0x76
ADDR+1	0x32	ADDR+1	0x54
ADDR+2	0x54	ADDR+2	0x32
ADDR+3	0x76	ADDR+3	0x10

Big  
Endian

00100

OK!

3<sup>rd</sup> Task: Try to send some text to the  
UDP echo server on tejo.tecnico.ulisboa.pt:58001.  
15 minutes.



```
//test.c
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <string.h>

int main(void)
{
    struct addrinfo hints,*res;
    int fd,errcode;
    ssize_t n;

    fd=socket(AF_INET,SOCK_DGRAM,0); //UDP socket
    if(fd==-1)/*error*/exit(1);

    memset(&hints,0,sizeof hints);
    hints.ai_family=AF_INET; //IPv4
    hints.ai_socktype=SOCK_DGRAM; //UDP socket

    errcode=getaddrinfo("tejo.tecnico.ulisboa.pt","58001",&hints,&res);
    if(errcode!=0)/*error*/exit(1);

    n=sendto(fd,"Hello!\n",7,0,res->ai_addr,res->ai_addrlen);
    if(n==-1)/*error*/exit(1);
    /*...*/
    freeaddrinfo(res);
    exit(0);
}
```

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain,int type,int protocol);
ssize_t sendto(int s,const void *buf,size_t len,int flags,
               const struct sockaddr *dest_addr,socklen_t addrlen);
```

```
struct addrinfo{           // (item in a linked list)
    int          ai_flags;   // additional options
    int          ai_family;  // address family
    int          ai_socktype; // socket type
    int          ai_protocol; // protocol
    socklen_t    ai_addrlen; // address length (bytes)
    struct sockaddr *ai_addr; // socket address
    char         *ai_canonname; // canonical hostname
    struct addrinfo *ai_next; // next item
};
```

```
struct sockaddr {
    unsigned short  sa_family; // address family
    char          sa_data[14]; // protocol specific address
};
```

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

int getaddrinfo(const char *node, const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);
```



last byte sent

More?    \$  
\$ man socket sendto getaddrinfo memset htons 7 ip

# UDP and recvfrom

```
//test.c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <string.h>
#include <unistd.h>

int main(void)
{
    int fd;
    struct sockaddr addr;
    socklen_t addrlen;
    ssize_t n;
    char buffer[128+1];

    /*...*/ //see previous task code

    addrlen = sizeof(addr);
    n = recvfrom(fd, buffer, 128, 0, &addr, &addrlen);
    if(n == -1) /*error*/ exit(1);
    buffer[n] = '\0';
    printf("echo: %s\n", buffer);

    close(fd);
    exit(0);
}
```

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t recvfrom(int s, void *buf, size_t len, int flags,
                 struct sockaddr *from, socklen_t *fromlen);
```

```
$ make
gcc test.c -o test
$ ./test
echo: Hello!
$
```

input/output argument

**Question 1:** What happens if the messages do not arrive at the destination? Try specifying a wrong port number for the destination echo server. Did you get an error message?

**Question 2:** How do you know the message you received came from the UDP echo server on tejo:58001.

**Question 3:** Which port number is your UDP client listening to when it is waiting for the echo reply?

**Question 4:** How many bytes do you expect to receive from recvfrom?

**Question 5:** Do you expect buffer content to be a NULL terminated string?

More? \$  
\$ man recvfrom

# Answers

**Answer to question 1:** No message will be received back at the client and it will block in recvfrom. No error will be detected unless timeouts are used.

You are using UDP. There are no guarantees that the messages will be delivered at the destination, and the order by which they are delivered may not be the same as that in which they were transmitted.

**Answer to question 2:** You have to check the recvfrom addr output argument. See, in the next slide, how to use getnameinfo for that purpose.

If you only want to receive messages from a specific address, then use send and recv. Find out more on manual page 2 (man 2 send recv).

**Answer to question 3:** The system assigned some unused port in the range when you first called sendto and this is the port recvfrom is listening to. If you want to use a specific port number you have to use bind. More on that later.

**Answer to question 4:** In this particular case, you should expect to receive 7 bytes (see sendto in previous slide).

**Answer to question 5:** In this particular case, you should not expect buffer to be NULL terminated. See sendto in previous slide and notice that the '\0' was not sent.

**Question 2:** How do you know the message you received came from the UDP echo server on tejo:58001.

**Question 3:** Which port number is your UDP client listening to when it is waiting for the echo reply?

**Question 4:** How many bytes do you expect to receive from recvfrom?

**Question 5:** Do you expect buffer content to be a NULL terminated string?

**Question 1:** What happens if the messages do not arrive at the destination? Try specifying a wrong port number for the destination echo server. Did you get an error message?

OK. Now let's move from UDP to TCP.

TCP is connection-oriented.

6<sup>th</sup> Task: Connect to the TCP echo server on [tejo.tecnico.ulisboa:58001](http://tejo.tecnico.ulisboa.pt:58001).

10 minutes.

# TCP, socket and connect

```
//test.c
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <string.h>

int main(void)
{
    struct addrinfo hints,*res;
    int fd,n;

    fd=socket(AF_INET,SOCK_STREAM,0); //TCP socket
    if(fd==-1)exit(1); //error

    memset(&hints,0,sizeof hints);
    hints.ai_family=AF_INET; //IPv4
    hints.ai_socktype=SOCK_STREAM; //TCP socket

    n=getaddrinfo("tejo.tecnico.ulisboa.pt","58001",&hints,&res);
    if(n!=0)/*error*/exit(1);

    n=connect(fd,res->ai_addr,res->ai_addrlen);
    if(n==-1)/*error*/exit(1);

    /*...*/
}
```

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

**Question 6:** Did you notice that the host name and port number are the same as before?

**Question 7:** What do you expect to happen if you type the wrong host name or port number?

Alternative:  
errcode=getaddrinfo("193.136.138.142","58001",&hints,&res);

More?

```
$  
$ man connect
```

# Answers

**Answer to question 6:** There is no problem in having two servers on the same port number as long as they are using different protocols. In this case, one is using UDP and the other TCP.

**Question 6:** Did you notice that the host name and port number are the same as before?

**Question 7:** What do you expect to happen if you type the wrong host name or port number?

**Answer to question 7:** If you type the wrong host name, `getaddrinfo` would give you an error, unless you type a name that also exists. If you type the wrong port number, `connect` would give you an error, unless there is a TCP server listening on that port.

7<sup>th</sup> Task: Send some text over the connection you have just established  
and read the response.  
10 minutes.

# TCP, write and read

```
//test.c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
/* ... */
int main(void)
{
    int fd;
    ssize_t nbytes,nleft,nwritten,nread;
    char *ptr,buffer[128+1];
    /*...*/ //see previous task code
    ptr=strcpy(buffer,"Hello!\n");
    nbytes=7;
    nleft=nbytes;
    while(nleft>0){nwritten=write(fd,ptr,nleft);
                  if(nwritten<=0)/*error*/exit(1);
                  nleft-=nwritten;
                  ptr+=nwritten;}
    nleft=nbytes; ptr=buffer;
    while(nleft>0){nread=read(fd,ptr,nleft);
                  if(nread== -1)/*error*/exit(1);
                  else if(nread==0)break;//closed by peer
                  nleft-=nread;
                  ptr+=nread;}
    nread=nbytes-nleft;

    buffer[nread] = '\0';
    printf("echo: %s\n", buffer);
    close(fd);
    exit(0);
}
```

```
#include <unistd.h>
ssize_t write(int fd,const void *buf,size_t count);
ssize_t read(int fd,void *buf,size_t count);
```

also used to write and  
read to/from files

```
$ make
gcc test.c -o test
$ ./test
echo: Hello!
$
```

**Question 8:** Did you notice that you may  
have to call write and read more than  
once?

**Question 9:** What do you expect to happen  
if your messages do not arrive at the  
destination?

More?

```
$
$ man 2 write read
```

# Answers

**Answer to question 8:** There is no guarantee that `write` would send all the bytes you requested when you called it. Transport layer buffers may be full. However, `write` returns the number of bytes that were sent (accepted by the transport layer). So, you just have to use this information to make sure everything is sent.

You may also have to call `read` more than once, since `read` would return as soon as data is available at the socket. It may happen that, when `read` returns, there was still data to arrive. Since `read` returns the number of bytes read from the socket, you just have to use this information to make sure nothing is missing.

**Answer to question 9:** If the transport layer can not deliver your messages to the destination, the connection will be lost. In some circumstances, this may take a few minutes due to timeouts. If your process is blocked in a `read` when the connection is lost, then `read` would return -1 and `errno` would be set to the appropriate error.

If you call `write` on a lost connection, `write` would return -1, `errno` will be set to EPIPE, but the system would raise a SIGPIPE signal and, by default, that would kill your process. See the next slide for a way to deal with the SIGPIPE signal.

Note however that, if the connection is closed, by the peer process, in an orderly fashion, while `read` is blocking your process, then `read` would return 0, as a sign of EOF(end-of-file).

**Question 8:** Did you notice that you may have to call `write` and `read` more than once?

**Question 9:** What do you expect to happen if your messages do not arrive at the destination?

Be careful. If the connection is lost and you write to the socket, the system will raise a SIGPIPE signal and, by default, this will kill your process.

8<sup>th</sup> Task: Protect the application against SIGPIPE signals.

5 minutes.

# TCP and the SIGPIPE signal

```
//test.c  
#include <signal.h>
```

```
/*...*/  
  
int main(void)  
{  
/*...*/  
    struct sigaction act;  
  
    memset(&act, 0, sizeof act);  
    act.sa_handler = SIG_IGN;  
  
    if (sigaction(SIGPIPE, &act, NULL) == -1) /*error*/ exit(1);  
/*...*/  
}
```

From now on, the SIGPIPE signal will be ignored.

```
#include <signal.h>  
int sigaction(int signum, const struct sigaction *act,  
             struct sigaction *oldact);
```

More?

```
$  
$ man sigaction 7 signal
```

Now, if the connection is lost and you write to the socket, the write will return -1 and errno will be set to EPIPE.

Let's move from clients to servers.

Servers have well-known ports.

9<sup>th</sup> Task: Write a UDP echo server and run it on port 58001.

15 minutes.

# UDP server and bind

```
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <string.h>

int main(void)
{
    struct addrinfo hints,*res;
    int fd,errcode;
    struct sockaddr addr;
    socklen_t addrlen;
    ssize_t n,nread;
    char buffer[128];

    if((fd=socket(AF_INET,SOCK_DGRAM,0))==-1)exit(1); //error

    memset(&hints,0,sizeof hints);
    hints.ai_family=AF_INET; //IPv4
    hints.ai_socktype=SOCK_DGRAM; //UDP socket
    hints.ai_flags=AI_PASSIVE;
    if((errcode=getaddrinfo(NULL,"58001",&hints,&res))!=0)/*error*/exit(1);

    if(bind(fd,res->ai_addr,res->ai_addrlen)==-1)/*error*/exit(1);

    while(1){addrlen=sizeof(addr);
        nread=recvfrom(fd,buffer,128,0, &addr,&addrlen);
        if(nread==-1)/*error*/exit(1);
        n=sendto(fd,buffer,nread,0,&addr,addrlen);
        if(n==-1)/*error*/exit(1);
    }
    //freeaddrinfo(res);
    //close(fd);
    //exit(0);
}
```

well-known  
port number

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

Use bind to register the server well known address (and port) with the system.

More?

```
$  
$ man 2 bind
```

**Question 10:** What do you expect to happen if there is already a UDP server on port 58001?

**Note:** You can also use bind to register the address (and port) in clients. In that case, if you set the port number to 0, the system assigns some unused port in the range 1024 through 5000.

Send only the bytes you read.

# Answers

**Question 10:** What do you expect to happen if there is already a UDP server on port 58001?

**Answer to question 10:** You would get an error on bind.

Now, do the same, but with TCP.

10<sup>th</sup> Task: Write a TCP echo server and run it also on port 58001.

20 minutes.

# TCP server, bind, listen and accept

```
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <string.h>
#include <unistd.h>

int main(void)
{
    struct addrinfo hints,*res;
    int fd,newfd,errcode;      ssize_t n,nw;
    struct sockaddr addr;      socklen_t addrlen;
    char *ptr,buffer[128];

    if((fd=socket(AF_INET,SOCK_STREAM,0))==-1)exit(1); //error

    memset(&hints,0,sizeof hints);
    hints.ai_family=AF_INET;//IPv4
    hints.ai_socktype=SOCK_STREAM;//TCP socket
    hints.ai_flags=AI_PASSIVE;
    if((errcode=getaddrinfo(NULL,"58001",&hints,&res))!=0)/*error*/exit(1);

    if(bind(fd,res->ai_addr,res->ai_addrlen)==-1)/*error*/exit(1);
    if(listen(fd,5)==-1)/*error*/exit(1);
```

```
        address of the connected peer process
```

```
    while(1){addrlen=sizeof(addr);
        if((newfd=accept(fd,&addr,&addrlen))==-1)
            /*error*/exit(1);
        while((n=read(newfd,buffer,128))!=0){if(n== -1)/*error*/exit(1);
            ptr=&buffer[0];
            while(n>0){if((nw=write(newfd,ptr,n))<=0)/*error*/exit(1);
                n-=nw; ptr+=nw;}}
        close(newfd);
    }
//freeaddrinfo(res);close(fd);exit(0);
}
```

Use bind to register the server well known address (and port) with the system.

Use listen to instruct the kernel to accept incoming connection requests for this socket. The backlog argument defines the maximum length the queue of pending connections may grow to.

Use accept to extract the first connection request on the queue of pending connections. Returns a socket associated with the new connection.

**Question 11:** Where do you expect the program to block?

**Question 12:** What happens if more than one client try to connect with the server?

**Note:** Do not forget to protect your application against the SIGPIPE signal.

More?

```
$  
$ man 2 bind listen accept 7 tcp
```

# Answers

**Answer to question 11:** This particular program is going to block in the accept call, until an incoming connection arrives. Then, it would block in the read call, until data is available at the newfd socket. Only after this connection is finished, the program would return to the accept call, where it would block if there are no pending connections waiting.

**Question 11:** Where do you expect the program to block?

**Answer to question 12:** As it was written, this program can only serve a client at a time. In the meantime, connections from other clients would become pending or would be rejected. The number of pending connections depends on the listen backlog argument.

**Question 12:** What happens if more than one client try to connect with the server?

If you are already serving a client, send “busy\n” to new incoming clients.

11<sup>th</sup> Task: Change the previous code to do that.

15 minutes.

# select

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
/* ... */
#define max(A,B) ((A)>=(B)?(A):(B))

int main(void)
{
    int fd,newfd,afd=0;
    fd_set rfds;
    enum {idle,busy} state;
    int maxfd,counter;
    /*...*/
    /*fd=socket(..);bind(fd,...);listen(fd,...);*/
    state=idle;
    while(1){FD_ZERO(&rfds);
        switch(state){
            case idle: FD_SET(fd,&rfds);maxfd=fd; break;
            case busy: FD_SET(fd,&rfds);FD_SET(afd,&rfds);maxfd=max(fd,afd); break;
        }//switch(state)

        counter=select(maxfd+1,&rfds,(fd_set*)NULL,(fd_set*)NULL,(struct timeval *)NULL);
        if(counter<=0)/*error*/exit(1);

        for(;counter;--counter)
            switch(state){
                case idle: if(FD_ISSET(fd,&rfds)){FD_CLR(fd,&rfds);
                    addrlen=sizeof(addr);
                    if((newfd=accept(fd,&addr,&addrlen))==-1)/*error*/exit(1);
                    afd=newfd;state=busy;}
                    break;
                case busy: if(FD_ISSET(fd,&rfds)){FD_CLR(fd,&rfds);
                    addrlen=sizeof(addr);
                    if((newfd=accept(fd,&addr,&addrlen))==-1)/*error*/exit(1);
                    /* ... write "busy\n" in newfd */
                    close(newfd);}
                    else if(FD_ISSET(afd,&rfds)){FD_CLR(afd,&rfds);
                        if((n=read(afd,buf,128))!=0)
                            {if(n== -1)/*error*/exit(1);
                            /* ... write buffer in afd */}
                        else{close(afd);state=idle;}//connection closed by peer
                    }
                    break;
            }//switch(state)
        }//while(1)
    /*close(fd);exit(0);*/
}//main
```

Returns the number of file descriptors ready.

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
int select(int n,fd_set *readfds,fd_set *writefds,
           fd_set *exceptfds,struct timeval *timeout);
FD_CLR(int fd,fd_set *set);
FD_ISSET(int fd,fd_set *set);
FD_SET(int fd,fd_set *set);
FD_ZERO(fd_set *set);
```

fd is ready

fd is ready

afd is ready

**Question 13:** And now, where do you expect the program to block?

\$  
\$ man 2 select

More?

# Answers

**Answer to question 13:** This program is only going to block in the `select` call. It would not block neither in the `accept` call, neither in the `read` call, since those are only executed when their sockets are ready to be read (and so they have no reason to block).

**Question 13:** And now, where do you expect the program to block?

# Further Reading

*Unix Network Programming: Networking APIs: Sockets and XTI (Volume I),*  
2<sup>nd</sup> ed., W. Richard Stevens, 1998, Prentice-Hall PTR, ISBN  
013490012X.

*Unix Network Programming: Networking APIs: The Sockets Networking API*  
(Volume I), 3<sup>rd</sup> ed., W. Richard Stevens, Bill Fenner, Andrew M. Rudoff,  
2003, Addison-Wesley Professional, ISBN 0131411551.