A Quick Guide to Networking Software

by J. Sanguino

2nd Edition: February 2010

Welcome, you are inside now.
1st Task: Get the host name!
You have 10 minutes.

gethostname

```
#include <unistd.h>
int gethostname(char *name,size_t len);
```

```
//test.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
extern int errno;

int main(void)
{
char buffer[128];

if(gethostname(buffer,128)==-1)
     printf("error: %s\n",strerror(errno));

else printf("host name: %s\n",buffer);
exit(0);
}
```

```
# makefile

test: test.c
<tab> gcc test.c -o test
```

running on tejo.ist.utl.pt

```
$ make
gcc test.c -o test
$ ./test
host name: tejo.ist.utl.pt
$
```

```
#include <string.h>

char *strerror(int errnum);
```

More?

```
$
$ man gethostname strerror
```

Good! Move on!
2nd Task: Now that you have a name, get the IP address.
15 minutes.

gethostbyname

```
#include <netdb.h>
struct hostent *gethostbyname(const char *name);
```

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

int main(void)
{
struct hostent *h;
struct in_addr *a;

if((h=gethostbyname("tejo"))==NULL)exit(1);//error

printf("official host name: %s\n",h->h_name);

a=(struct in_addr*)h->h_addr_list[0];
printf("internet address: %s (%08lX)\n",inet_ntoa(*a),ntohl(a->s_addr));
exit(0);
}
```

```
struct hostent{
char *h_name;       // official host name
char **h_aliases;   // alias list
int  h_addrtype;    // host address type
int  h_length;      // length of address
char **h_addr_list; // list of addresses (NULL term.)
};
```

```
struct in_addr{
u_int32_t      s_addr; // 32 bits
};
```

0xC0==192
0xA8==168

```
$ make
gcc test.c -o test
$ ./test
official host name: tejo.ist.utl.pt
internet address: 192.168.0.1 (C0A80001)
$
```

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

char *inet_ntoa(struct in_addr in);
```

```
#include <arpa/inet.h>
uint32_t ntohl(uint16_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

More?

```
$
$ man gethostbyname inet_ntoa 7 ip
```

OK!
3rd Task: Try to send some text to the UDP echo server on tejo:58000.
15 minutes.

UDP, socket and sendto

host name

port number

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

int main(void)
{
int fd, n;
struct sockaddr_in addr;
/* ... */
fd=socket(AF_INET,SOCK_DGRAM,0);//UDP socket
if(fd==-1)exit(1);//error

memset((void*)&addr,(int)'\0',sizeof(addr));
addr.sin_family=AF_INET;
addr.sin_addr.s_addr=
addr.sin_port=htons(58000);

n=sendto(fd,"Hello!\n",7,0,(struct sockaddr*)&addr,sizeof(addr));
if(n==-1)exit(1);//error
/* ... */
}
```

Put the server IP address here.
Use gethostbyname.

```c
#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 *to,socklen_t tolen);
```

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

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

```c
#include <arpa/inet.h>
uint16_t htons(uint16_t hostshort);
       (host to network short)
```

'H'	'e'	'l'	'l'	'o'	'!'	'\n'	'\0'

last byte sent

```
Short (16 bits) 0x3210
Little endian system:
                ADDR   0x10
                ADDR+1 0x32
Network byte order:
(Big Endian)    ADDR   0x32
                ADDR+1 0x10
```

More?

```
$
$ man socket sendto memset htons 7 ip
```

4th Task: Now, receive the echo from the UDP echo server.
20 minutes.

UDP and recvfrom

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

int main(void)
{
int fd, n, addrlen;
struct sockaddr_in addr;
char buffer[128];

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

addrlen=sizeof(addr);
n=recvfrom(fd,buffer,128,0,(struct sockaddr*)&addr,&addrlen);
if(n==-1)exit(1);//error

write(1,"echo: ",6);//stdout
write(1,buffer,n);

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

```c
#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 2: How do you know the message you received came from the UDP echo server on `tejo:58000`.

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

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 got an error message?

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

Question 5: Do you expect `buffer` 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 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 `gethostbyaddr` 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 1024 through 5000 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 transmitted.

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 got an error message?

Question 2: How do you know the message you received came from the UDP echo server on `tejo:58000`.

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` to be a `NULL` terminated string?

5th Task: Check who sent you the message.
10 minutes.

gethostbyaddr

```
#include <netdb.h>
#include <sys/socket.h> /* for AF_INET */
struct hostent *gethostbyaddr(const void *addr,int len,int type);
```

```
//test.c
#include <stdio.h>
#include <netdb.h>
#include <sys/socket.h>
/* ... */

int main(void)
{
int fd, n, addrlen;
struct sockaddr_in addr;
char buffer[128];

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

addrlen=sizeof(addr);
n=recvfrom(fd,buffer,128,0,(struct sockaddr*)&addr,&addrlen);
if(n==-1)exit(1);//error
/* ... */

h=gethostbyaddr((char*)&addr.sin_addr,sizeof(struct in_addr),AF_INET);
if(h==NULL)
     printf("sent by [%s:%hu]\n",inet_ntoa(addr.sin_addr),ntohs(addr.sin_port));
else printf("sent by [%s:%hu]\n",h->h_name,ntohs(addr.sin_port));

exit(0);
}
```

output argument

```
$ make
gcc test.c -o test
$ ./test
echo: Hello!
sent by [tejo.ist.utl.pt:8001]
$
```

```
#include <arpa/inet.h>
uint16_t ntohs(uint16_t netshort);
      (network to host short)
```

More?

```
$
$ man gethostbyaddr
```

OK. Now let's move from UDP to TCP.
TCP is connection-oriented.
6th Task: Connect to the TCP echo server on tejo:58000.
10 minutes.

TCP, socket and connect

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

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

int main(void)
{
int fd, n;
struct sockaddr_in addr;
/* ... */
fd=socket(AF_INET,SOCK_STREAM,0);//TCP socket
if(fd==-1)exit(1);//error

memset((void*)&addr,(int)'\0',sizeof(addr));
addr.sin_family=AF_INET;
addr.sin_addr.s_addr=
addr.sin_port=htons(58000);

n=connect(fd,(struct sockaddr*)&addr,sizeof(addr));
if(n==-1)exit(1);//error
/* ... */
}
```

As before, put the server IP address here.

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?

More?

```
$
$ man connect
```

Answers

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

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 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, `gethostbyname` 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.

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

TCP, write and read

```c
//test.c
#include <unistd.h>
#include <string.h>
/* ... */
int main(void)
{
int fd, nbytes, nleft, nwritten, nread;
char *ptr, buffer[128];
/* ... *// see previous task code
ptr=strcpy(buffer,"Hello!\n");
nbytes=7;

nleft=nbytes;
while(nleft>0){nwritten=write(fd,ptr,nleft);
               if(nwritten<=0)exit(1);//error
               nleft-=nwritten;
               ptr+=nwritten;}
nleft=nbytes; ptr=&buffer[0];
while(nleft>0){nread=read(fd,ptr,nleft);
               if(nread==-1)exit(1);//error
               else if(nread==0)break;//closed by peer
               nleft-=nread;
               ptr+=nread;}
nread=nbytes-nleft;
close(fd);

write(1,"echo: ",6);//stdout
write(1,buffer,nread);
exit(0);
}
```

```c
#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 that 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.
8th Task: Protect the application against SIGPIPE signals.
5 minutes.

TCP and the SIGPIPE signal

```c
#include <signal.h>
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

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

/* ... */

int main(void)
{
void (*old_handler)(int);//interrupt handler

/* ... */

if((old_handler=signal(SIGPIPE,SIG_IGN))==SIG_ERR)exit(1);//error

/* ... */

}
```

From now on, the SIGPIPE signal will be ignored.

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

More?

```
$
$ man 2 signal 7 signal
```

Let's move from clients to servers.
Servers have well-known ports.
9th Task: Write a UDP echo server and run it on port 59000.
15 minutes.

UDP server and bind

```c
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>

int main(void)
{
int fd, addrlen, ret, nread;
struct sockaddr_in addr;
char buffer[128];

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

memset((void*)&addr,(int)'\0',sizeof(addr));
addr.sin_family=AF_INET;
addr.sin_addr.s_addr=htonl(INADDR_ANY);
addr.sin_port=htons(59000);

ret=bind(fd,(struct sockaddr*)&addr,sizeof(addr));
if(ret==-1)exit(1);//error

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

well-known port number

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

Accept datagrams on any Internet interface on the system.

Send only the bytes you read.

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

More?

```
$
$ man 2 bind
```

Question 10: What do you expect to happen if there is already a UDP server on port 59000?

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.

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

Answers

Question 10: What do you expect to happen if there is already a UDP server on port 59000?

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

Now, do the same, but with TCP.
10th Task: Write a TCP echo server and run it also on port 59000.
20 minutes.

TCP server, bind, listen and accept

```c
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>

int main(void)
{
int fd, addrlen, newfd;
struct sockaddr_in addr;
int n, nw;
char *ptr, buffer[128];

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

memset((void*)&addr,(int)'\0',sizeof(addr));
addr.sin_family=AF_INET;
addr.sin_addr.s_addr=htonl(INADDR_ANY);
addr.sin_port=htons(59000);
if(bind(fd,(struct sockaddr*)&addr,sizeof(addr))==-1)
    exit(1);//error

if(listen(fd,5)==-1)exit(1);//error

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

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

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.

address of the connected peer process

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.
11th Task: Change the previous code to do that.
15 minutes.

select

```c
#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;
fd_set rfds;
enum {idle,busy} state;
int maxfd, counter;
/* ... */
/* fd=socket(…); bind(fd,…); listen(fd,…) */
state=idle;
while(1){FD_ZERO(&rfds);
         FD_SET(fd,&rfds);maxfd=fd;
         if(state==busy){FD_SET(afd,&rfds);maxfd=max(maxfd,afd);}

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

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

```c
#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);
```

Blocks until one of the file descriptors, previously set in rfds, are ready to by read.

Returns the number of file descriptors ready.

fd is ready

afd is ready

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

More?

```
$
$ man 2 select
```

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?

12th Task: Make your server a concurrent server.
15 minutes.

fork

```c
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
#include <errno.h>
extern int errno;

int main(void)
{
int fd, newfd, addrlen, n, nw, ret;
struct sockaddr_in addr;
char *ptr, buffer[128];
pid_t pid;
void (*old_handler)(int);//interrupt handler

if((old_handler=signal(SIGCHLD,SIG_IGN))==SIG_ERR)exit(1);//error

if((fd=socket(AF_INET,SOCK_STREAM,0))==-1)exit(1);//error
memset((void*)&addr,(int)'\0',sizeof(addr));
addr.sin_family=AF_INET;
addr.sin_addr.s_addr=htonl(INADDR_ANY);
addr.sin_port=htons(9000);
if(bind(fd,(struct sockaddr*)&addr,sizeof(addr))==-1)exit(1);//error
if(listen(fd,2)==-1)exit(1);//error

while(1){addrlen=sizeof(addr);
         do newfd=accept(fd,(struct sockaddr*)&addr,&addrlen);//wait for a connection
         while(newfd==-1&&errno==EINTR);
         if(newfd==-1)exit(1);//error

         if((pid=fork())==-1)exit(1);//error
         else if(pid==0)//child process
            {close(fd);
             while((n=read(newfd,buffer,128))!=0){if(n==-1)exit(1);//error
                 ptr=&buffer[0];
                 while(n>0){if((nw=write(newfd,ptr,n))<=0)exit(1);//error
                            n-=nw; ptr+=nw;}
                 }
             close(newfd); exit(0);}
         //parent process
         do ret=close(newfd);while(ret==-1&&errno==EINTR);
         if(ret==-1)exit(1);//error
        }
/* close(fd); exit(0); */}
```

Use fork to create a new process for each new connection.

```c
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

Avoid zombies when child processes die.

Note: Do not forget to protect the child process against the SIGPIPE signal.

Create a child process for each new connection.

Parent process may be interrupted by SIG_CHLD signal (child process death).

child process

More?

```
$
$ man 2 fork
```

Further Reading

Unix Network Programming: Networking APIs: Sockets and XTI (Volume 1), 2nd ed., W. Richard Stevens, 1998, Prentice-Hall PTR, ISBN 013490012X.

Unix Network Programming: Networking APIs: The Sockets Networking API (Volume 1), 3rd ed., W. Richard Stevens, Bill Fenner, Andrew M. Rudoff, 2003, Addison-Wesley Professional, ISBN 0131411551.