Linux Networking: socket programming, basis for services

David Morgan

Sockets

- a communications interface/mechanism
- like IPC (inter-process communications)
- but generalized to span machines
  (inter-*machine* inter-process)
- coded like file handles

Comparison: file handling

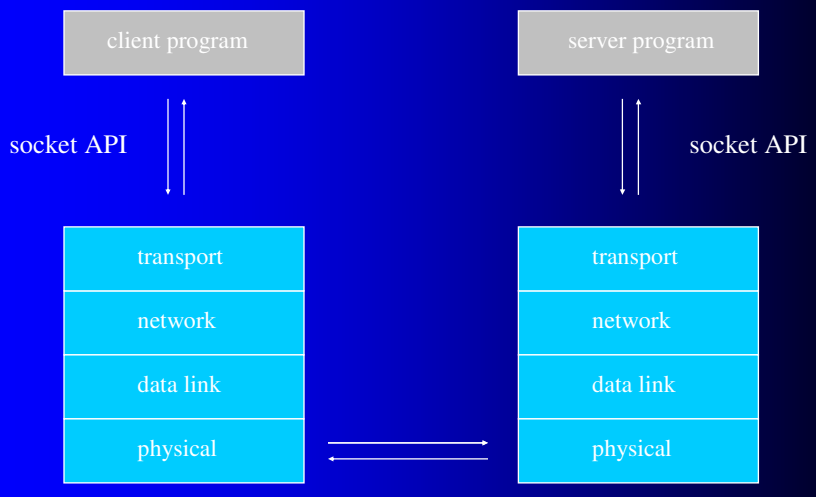
```
[root@frausto ~]# echo hello > file.in
[root@frausto ~]# ls file.*                          <-- file.in exists, ontains "hello"
file.in
[root@frausto ~]# cat -n file-operations.c
     1  #include <unistd.h>
     2  #include <sys/stat.h>
     3  #include <fcntl.h>
     4  #include <stdio.h>
     5
     6  int main()
     7  {
     8      char c;
     9      int in, out;
    10
    11      /* a set of OS-provided service functions handle files for you */
    12
    13      /* open files */
    14      in = open("file.in", O_RDONLY);
    15      out = open("file.out", O_WRONLY|O_CREAT, S_IRUSR|S_IWUSR);
    16
    17      /* read and write files */
    18      while(read(in,&c,1) == 1)
    19          write(out,&c,1);
    20
    21      /* close files */
    22      close(out);
    23  }
[root@frausto ~]# gcc file-operations.c -o file-operations
[root@frausto ~]# ./file-operations
[root@frausto ~]# ls file.*
file.in  file.out
[root@frausto ~]# cat file.out
hello                                                <-- file.out exists, ontains "hello"
```

A program that copies source file "file.in" to destination file "file.out"

Provided functions do the job

- open ( ) in lines 14 and 15
- read ( ) in line 18
- write ( ) in line 19
- close ( ) in line 22

Socket API



Server socket operations

- create a socket
- name it
- listen/queue incoming connections
- make another socket per queued connection
- use the socket(s) (send & receive)

Client socket operations

- create a socket
- connect it to a (presumed) server's socket
- use the socket (send & receive)

Socket system calls - servers

- socket( ) – to create
- bind( ) – to name
- listen( ) – to detect incoming connections
- accept( ) – to make a separate socket for each
- read( ), recv( ) ; write( ), send( ) – to use

Socket system calls - clients

- socket( ) – to create
- connect( ) – to reach out to a server's socket somewhere
- read( ), recv( ) ; write( ), send( ) – to use

Server program template

```
socket( )
bind( identifier to socket )

listen( )
while {
        accept( )

        recv( )
        send( )
        }
```


Client program template

```
socket( )
connect( to presumed server socket )
send( )
recv( )
```

Example: a client

```
 1  #include <sys/types.h>
 2  #include <sys/socket.h>
 3  #include <stdio.h>
 4  #include <netinet/in.h>
 5  #include <arpa/inet.h>
 6  #include <unistd.h>
 7  int main()
 8  {
 9     int sockfd;
10     int len;
11     struct sockaddr_in address;
12     int result;
13     char ch = 'R';
14
15     sockfd = socket(AF_INET, SOCK_STREAM, 0);
16
17     address.sin_family = AF_INET;
18     address.sin_addr.s_addr = inet_addr("192.168.1.1");
19     address.sin_port = htons(819);
20     len = sizeof(address);
21
22     result = connect(sockfd, (struct sockaddr *)&address, len);
23
24     if(result == -1) {
25        perror("oops: client3");
26        exit(1);
27     }
28
29     write(sockfd, &ch, 1);
30     read(sockfd, &ch, 1);
31     printf("char from server = %c\n", ch);
32     close(sockfd);
33     exit(0);
34  }
```

A program that sends a letter of the alphabet to another program
(namely, 192.168.1.1 port 819)

Provided functions do the job

- socket ( ) in line 15
- connect ( ) in line 22
- write ( ) in line 29
- read ( ) in line 30
- close ( ) in line 32

Example: a server

```
 1  #include <sys/types.h>
 2  #include <sys/socket.h>
 3  #include <stdio.h>
 4  #include <netinet/in.h>
 5  #include <arpa/inet.h>
 6  #include <unistd.h>
 7  int main()
 8  {
 9     int server_sockfd, client_sockfd;
10     int server_len, client_len;
11     struct sockaddr_in server_address;
12     struct sockaddr_in client_address;
13
14     server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
15
16     server_address.sin_family = AF_INET;
17     server_address.sin_addr.s_addr = htonl(INADDR_ANY);
18     server_address.sin_port = htons(9999);
19     server_len = sizeof(server_address);
20     bind(server_sockfd, (struct sockaddr *)&server_address, server_len);
21     listen(server_sockfd, 5);
22     while(1) {
23        char ch;
24        printf("server waiting\n");
25        client_len = sizeof(client_address);
26        client_sockfd = accept(server_sockfd,
27           (struct sockaddr *)&client_address, &client_len);
28
29        read(client_sockfd, &ch, 1);
30        ch++;
31        write(client_sockfd, &ch, 1);
32        close(client_sockfd);
33     }
34  }
```

A program that, on getting a letter of the alphabet from its client, send the next letter to the client

Provided functions do the network communication job

- socket ( ) in line 14
- bind ( ) in line 20
- listen ( ) in line 21
- accept ( ) in line 26
- read ( ) in line 29
- write ( ) in line 31
- close ( ) in line 32

While programmer's logic does the specific job that is the program's purpose, reacting to the received after its receipt and before sending of the response (line 30)

Socket naming

- uses a C structure
- primary elements: an IP addr & port number

Sockets produced on server

- server's named socket for connection setup
- separate individual unnamed sockets spawned per conversation thereafter

Socket references

Computer Networks and Internets, Douglas Comer, Prentice Hall, 2001 Chapter 27, "The Socket Interface," and

Beginning Linux Progamming, Neil Matthew and Richard Stones, Wrox Press, 1996, Chapter 13, "Sockets."

Linux Programming Unleashed, Wall, Watson, and Whitis, SAMS, 1999, Chapter 19, "TCP/IP and Socket Programming"

http://homepage.smc.edu/morgan_david/cs70/sockets.html

http://www.ods.com.ua/win/eng/rfc/socket/