| Winsock & .NET | Winsock | < Iterative Server Program Example | Linux Socket Index | The TCP/IP Stack > |

NETWORK PROGRAMMING LINUX SOCKET PART 13: MULTICAST

Working program examples if any compiled using gcc, tested using the public IPs, run on **Linux Fedora 3** with several times update, as normal user. The Fedora machine used for the testing having the "No Stack Execute" disabled and the SELinux set to default configuration. All the program example is generic. Beware codes that expand more than one line.

Example: Sending and receiving a multicast datagram

- IP multicasting provides the capability for an application to send a single IP datagram that a group of hosts in a network can receive. The hosts that are in the group may reside on a single subnet or may be on different subnets that have been connected by multicast capable routers.
- Hosts may join and leave groups at any time. There are no restrictions on the location or number of members in a host group. A class D Internet address in the range 224.0.0.1 to 239.255.255.255 identifies a host group.
- An application program can send or receive multicast datagrams by using the socket() API and connectionless SOCK_DGRAM type sockets. Each multicast transmission is sent from a single network interface, even if the host has more than one multicasting-capable interface.
- It is a one-to-many transmission method. You cannot use connection-oriented sockets of type SOCK_STREAM for multicasting.

- When a socket of type SOCK_DGRAM is created, an application can use the setsockopt() function to control the multicast characteristics associated with that socket. The setsockopt() function accepts the following IPPROTO_IP level flags:

  1. IP_ADD_MEMBERSHIP: Joins the multicast group specified.
  2. IP_DROP_MEMBERSHIP: Leaves the multicast group specified.
  3. IP_MULTICAST_IF: Sets the interface over which outgoing multicast datagrams are sent.
  4. IP_MULTICAST_TTL: Sets the Time To Live (TTL) in the IP header for outgoing multicast datagrams. By default it is set to 1. TTL of 0 are not transmitted on any sub-network. Multicast datagrams with a TTL of greater than 1 may be delivered to more than one sub-network, if there are one or more multicast routers attached to the first sub-network.
  5. IP_MULTICAST_LOOP: Specifies whether or not a copy of an outgoing multicast datagram is delivered to the sending host as long as it is a member of the multicast group.

- The following examples enable a socket to send and receive multicast datagrams. The steps needed to send a multicast datagram differ from the steps needed to receive a multicast datagram.

Example: Sending a multicast datagram, a server program

- The following example enables a socket to perform the steps listed below and to send multicast datagrams:

  1. Create an AF_INET, SOCK_DGRAM type socket.
  2. Initialize a sockaddr_in structure with the destination group IP address and port number.
  3. Set the IP_MULTICAST_LOOP socket option according to whether the sending system should receive a copy of the multicast datagrams that are transmitted.
  4. Set the IP_MULTICAST_IF socket option to define the local interface over which you want to send the multicast datagrams.
  5. Send the datagram.

```
[bodo@bakawali testsocket]$ cat
mcastserver.c
```

```
/* Send Multicast Datagram code example. */
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>


struct in_addr localInterface;
struct sockaddr_in groupSock;
int sd;
char databuf[1024] = "Multicast test message lol!";
int datalen = sizeof(databuf);

int main (int argc, char *argv[ ])
{
/* Create a datagram socket on which to send. */
sd = socket(AF_INET, SOCK_DGRAM, 0);
if(sd < 0)
{
  perror("Opening datagram socket error");
  exit(1);
}
else
  printf("Opening the datagram socket...OK.\n");

/* Initialize the group sockaddr structure with a */
/* group address of 225.1.1.1 and port 5555. */
memset((char *) &groupSock, 0, sizeof(groupSock));
groupSock.sin_family = AF_INET;
groupSock.sin_addr.s_addr = inet_addr("226.1.1.1");
groupSock.sin_port = htons(4321);

/* Disable loopback so you do not receive your own
datagrams.
{
char loopch = 0;
if(setsockopt(sd, IPPROTO_IP, IP_MULTICAST_LOOP,
(char *)&loopch, sizeof(loopch)) < 0)
{
perror("Setting IP_MULTICAST_LOOP error");
close(sd);
exit(1);
}
else
printf("Disabling the loopback...OK.\n");
```

```
}
*/

/* Set local interface for outbound multicast
datagrams. */
/* The IP address specified must be associated with a
local, */
/* multicast capable interface. */
localInterface.s_addr = inet_addr("203.106.93.94");
if(setsockopt(sd, IPPROTO_IP, IP_MULTICAST_IF, (char
*)&localInterface, sizeof(localInterface)) < 0)
{
  perror("Setting local interface error");
  exit(1);
}
else
  printf("Setting the local interface...OK\n");
/* Send a message to the multicast group specified by
the*/
/* groupSock sockaddr structure. */
/*int datalen = 1024;*/
if(sendto(sd, databuf, datalen, 0, (struct
sockaddr*)&groupSock, sizeof(groupSock)) < 0)
{perror("Sending datagram message error");}
else
  printf("Sending datagram message...OK\n");

/* Try the re-read from the socket if the loopback is
not disable
if(read(sd, databuf, datalen) < 0)
{
perror("Reading datagram message error\n");
close(sd);
exit(1);
}
else
{
printf("Reading datagram message from
client...OK\n");
printf("The message is: %s\n", databuf);
}
*/
return 0;
}
```

- Compile and link the program.

```
[bodo@bakawali testsocket]$ gcc -g mcastserver.c -o
mcastserver
```

- Before running this multicaster program, you have to run the client program first as in the following.

Example: Receiving a multicast datagram, a client

- The following example enables a socket to perform the steps listed below and to receive multicast datagrams:

1. Create an AF_INET, SOCK_DGRAM type socket.
2. Set the SO_REUSEADDR option to allow multiple applications to receive datagrams that are destined to the same local port number.
3. Use the bind() verb to specify the local port number. Specify the IP address as INADDR_ANY in order to receive datagrams that are addressed to a multicast group.
4. Use the IP_ADD_MEMBERSHIP socket option to join the multicast group that receives the datagrams. When joining a group, specify the class D group address along with the IP address of a local interface. The system must call the IP_ADD_MEMBERSHIP socket option for each local interface receiving the multicast datagrams.
5. Receive the datagram.

```
/* Receiver/client multicast Datagram example. */
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>

struct sockaddr_in localSock;
struct ip_mreq group;
int sd;
int datalen;
char databuf[1024];

int main(int argc, char *argv[])
{
/* Create a datagram socket on which to receive. */
sd = socket(AF_INET, SOCK_DGRAM, 0);
if(sd < 0)
{
perror("Opening datagram socket error");
```

```
exit(1);
}
else
printf("Opening datagram socket....OK.\n");

/* Enable SO_REUSEADDR to allow multiple instances of
this */
/* application to receive copies of the multicast
datagrams. */
{
int reuse = 1;
if(setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, (char
*)&reuse, sizeof(reuse)) < 0)
{
perror("Setting SO_REUSEADDR error");
close(sd);
exit(1);
}
else
printf("Setting SO_REUSEADDR...OK.\n");
}

/* Bind to the proper port number with the IP address
*/
/* specified as INADDR_ANY. */
memset((char *) &localSock, 0, sizeof(localSock));
localSock.sin_family = AF_INET;
localSock.sin_port = htons(4321);
localSock.sin_addr.s_addr = INADDR_ANY;
if(bind(sd, (struct sockaddr*)&localSock,
sizeof(localSock)))
{
perror("Binding datagram socket error");
close(sd);
exit(1);
}
else
printf("Binding datagram socket...OK.\n");

/* Join the multicast group 226.1.1.1 on the local
203.106.93.94 */
/* interface. Note that this IP_ADD_MEMBERSHIP option
must be */
/* called for each local interface over which the
multicast */
/* datagrams are to be received. */
group.imr_multiaddr.s_addr = inet_addr("226.1.1.1");
```

```
group.imr_interface.s_addr =
inet_addr("203.106.93.94");
if(setsockopt(sd, IPPROTO_IP, IP_ADD_MEMBERSHIP,
(char *)&group, sizeof(group)) < 0)
{
perror("Adding multicast group error");
close(sd);
exit(1);
}
else
printf("Adding multicast group...OK.\n");

/* Read from the socket. */
datalen = sizeof(databuf);
if(read(sd, databuf, datalen) < 0)
{
perror("Reading datagram message error");
close(sd);
exit(1);
}
else
{
printf("Reading datagram message...OK.\n");
printf("The message from multicast server is:
\"%s\"\n", databuf);
}
return 0;
}
```

- Compile and link.

```
[bodo@bakawali testsocket]$ gcc -g  mcastclient.c -o
mcastclient
```

- Run the client program.

```
[bodo@bakawali testsocket]$ ./mcastclient
Opening datagram socket....OK.
Setting SO_REUSEADDR...OK.
Binding datagram socket...OK.
Adding multicast group...OK.
```

- Then run the server program.

```
[bodo@bakawali testsocket]$ ./mcastserver
Opening the datagram socket...OK.
Setting the local interface...OK
Sending datagram message...OK
[bodo@bakawali testsocket]$
```

- The messages on the client console are shown below.

```
[bodo@bakawali testsocket]$ ./mcastclient
Opening datagram socket....OK.
Setting SO_REUSEADDR...OK.
Binding datagram socket...OK.
Adding multicast group...OK.
Reading datagram message...OK.
The message from multicast server is: "Multicast test
message lol!"
[bodo@bakawali testsocket]$
```

Continue on next Module…TCP/IP and RAW socket, more program examples.

Further reading and digging:

1. Check the best selling C/C++, Networking, Linux and Open Source books at Amazon.com.

2. Broadcasting.
3. Telephony.
4. GCC, GDB and other related tools.

| Winsock & .NET | Winsock | < Iterative Server Program Example | Linux Socket Index | The TCP/IP Stack > |