CpSc 360: Distributed and Network Programming

Broadcast & Multicast Programming

James Wang

Chapter 20, 21

Broadcast Programming

- **Broadcast Addresses:**
  - Subnet-directed broadcast address: {subnetid, -1} where subnetid represents the bits that are covered by the network mask (or the CIDR prefix).
  - limited broadcast address: {-1,-1,-1} or 255.255.255.255. It is normally used as the destination address during bootstrap process by applications such as BOOTP and DHCP, which do not yet know the node's IP addresse.
- **Program options:**
  - Set socket option to SO_BROADCAST, otherwise the kernel does not know it is a broadcasting packet.

2

A Broadcast Send Program

```
#include <stdio.h>     /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket() and bind() */
#include <arpa/inet.h>  /* for sockaddr_in */
#include <stdlib.h>    /* for atoi() and exit() */
#include <string.h>    /* for memset() */
#include <unistd.h>    /* for close() */

void ExitWithError(char *errorMessage);  /* External error handling function */

int main(int argc, char *argv[])
{
    int sock;                        /* Socket */
    struct sockaddr_in broadcastAddr; /* Broadcast address */
    char *broadcastIP;               /* IP broadcast address */
    unsigned short broadcastPort;    /* Server port */
    char *sendString;                /* String to broadcast */
    int broadcastPermission;         /* Socket opt to set permission to broadcast */
    unsigned int sendStringLen;      /* Length of string to broadcast */

    if (argc < 4)                    /* Test for correct number of parameters */
    {
        fprintf(stderr,"Usage:  %s <IP Address> <Port> <Send String>\n", argv[0]);
        exit(1);
    }

    broadcastIP = argv[1];           /* First arg:  broadcast IP address */
    broadcastPort = atoi(argv[2]);   /* Second arg:  broadcast port */
    sendString = argv[3];            /* Third arg:  string to broadcast */

    /* Create socket for sending/receiving  datagrams */
    if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
        ExitWithError("socket() failed");
```

3

A Broadcast Send Program (cont.)

```
    /* Set socket to allow broadcast */
    broadcastPermission = 1;
    if (setsockopt(sock, SOL_SOCKET, SO_BROADCAST, (void *) &broadcastPermission,
          sizeof(broadcastPermission)) < 0)
        ExitWithError("setsockopt() failed");

    /* Construct local address structure */
    memset(&broadcastAddr, 0, sizeof(broadcastAddr));   /* Zero out structure */
    broadcastAddr.sin_family = AF_INET;                 /* Internet address family */
    broadcastAddr.sin_addr.s_addr = inet_addr(broadcastIP);/* Broadcast IP address */
    broadcastAddr.sin_port = htons(broadcastPort);      /* Broadcast  port */

    sendStringLen = strlen(sendString);  /* Find length of sendString */
    for (;;) /* Run forever */
    {
        /* Broadcast sendString in datagram to clients  every 3 seconds*/
        if (sendto(sock, sendString, sendStringLen, 0, (struct sockaddr *)
                &broadcastAddr, sizeof(broadcastAddr)) != sendStringLen)
            ExitWithError("sendto() sent a different number of bytes than expected");

        sleep(3);   /* Avoids flooding the network */
    }
    /* NOT REACHED */
}
void ExitWithError(char *errorMessage)
{
    perror(errorMessage);
    exit(1);
}
```

4

A Broadcast Receive Program

```
#include <stdio.h>      /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket(), connect(), sendto(), and recvfrom() */
#include <arpa/inet.h>  /* for sockaddr_in and inet_addr() */
#include <stdlib.h>     /* for atoi() and exit() */
#include <string.h>     /* for memset() */
#include <unistd.h>     /* for close() */

#define MAXRECVSTRING 255  /* Longest string to receive */

void ExitWithError(char *errorMessage);  /* External error  handling function */

int main(int argc, char *argv[])
{
    int sock;                        /* Socket */
    struct sockaddr_in broadcastAddr; /* Broadcast Address */
    unsigned short broadcastPort;    /* Port */
    char recvString[MAXRECVSTRING+1]; /* Buffer for received string */
    int recvStringLen;               /* Length of received string */

    if (argc != 2)    /* Test for correct number of arguments */
    {
        fprintf(stderr,"Usage: %s <Broadcast Port>\n", argv[0]);
        exit(1);
    }

    broadcastPort = atoi(argv[1]);   /* First arg: broadcast port */
```

5

A Broadcast Receive Program (cont.)

```
    /* Create a best-effort datagram  socket using UDP */
    if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
        ExitWithError("socket() failed");

    /* Construct bind structure */
    memset(&broadcastAddr, 0, sizeof(broadcastAddr));   /* Zero out structure */
    broadcastAddr.sin_family = AF_INET;                 /* Internet address family */
    broadcastAddr.sin_addr.s_addr = htonl(INADDR_ANY);  /* Any incoming interface */
    broadcastAddr.sin_port = htons(broadcastPort);      /* Broadcast  port */

    /* Bind to the broadcast  port */
    if (bind(sock, (struct  sockaddr *) &broadcastAddr,  sizeof(broadcastAddr)) < 0)
        ExitWithError("bind() failed");

    /* Receive a single datagram from the server */
    if ((recvStringLen = recvfrom(sock, recvString,  MAXRECVSTRING, 0, NULL, 0)) < 0)
        ExitWithError("recvfrom() failed");

    recvString[recvStringLen] = '\0';
    printf("Received:  %s\n", recvString);    /* Print the received string */

    close(sock);
    exit(0);
}
void ExitWithError(char *errorMessage)
{
    perror(errorMessage);
    exit(1);
}
```

6

Multicast Programming

- **Addresses: Class D addresses, in range 224.0.0.0 through 239.255.255.255, are the multicast addresses in IPv4.**
- **Special IPv4 multicast addresses:**
  - **224.0.0.1:** all-hosts group. All multicast-capable nodes on a subnet must join this group on all multicast-capable interfaces.
  - **224.0.0.2:** all-routers group. All multicast-capable routers on a subnet must join this group on all multicast capable interfaces.

7

Scope of Multicast Addresses

Scope	IPv6 scope	IPv4	
		TTL Scope	Admin. Scope
Interface-local	1	0	
Link-local	2	1	224.0.0.0 to 224.0.0.255
Site-local	5	<32	239.255.0.0 to 239.255.255.255
Organization-local	8		239.192.0.0 to 239.195.255.255
Global	14	≤255	224.0.1.0 to 238.255.255.255

8

Multicast Socket Options

Sockets API Symbol	Description
IP_ADD_MEMBERSHIP	Adds the host to the membership of a multicast group. A host must become a member of a multicast group before it can receive datagrams sent to the group. Membership is associated with a single interface; programs running on multihomed hosts may need to join the same group on more than one interface. Up to IP_MAX_MEMBERSHIPS (currently 20) memberships may be added on a single socket.
IP_DROP_MEMBERSHIP	Removes the host from the membership of a multicast group.
IP_HDRINCL	If specified for a raw IP socket, you must build the IP header for all datagrams sent on the raw socket.
IP_MULTICAST_IF	Specifies the interface for outgoing multicast datagrams sent on this socket. The interface is specified as an `in_addr` structure.
IP_MULTICAST_LOOP	Disables loopback of local delivery. If a multicast datagram is sent to a group which the sending host is a member, a copy of the datagram is looped back by the IP layer for local delivery (the default). To disable the loopback delivery, specify a value of 0.
IP_MULTICAST_TTL	Specifies the time-to-live (TTL) value for outgoing multicast datagrams. Takes an integer value between 0 and 255.
IP_OPTIONS	Provides IP options to be transmitted in the IP header of each outgoing packet.
IP_RECVDSTADDR	Enables a SOCK_DGRAM socket to receive the destination IP address for a UDP datagram.
IP_RECVOPTS	Enables a SOCK_DGRAM socket to receive IP options.
IP_TTL	Time to live (TTL) for a datagram.
IP_TOS	Type of service (1-byte value).

9

Using TTL to Specify Scope

Value	Action
0	Restricts distribution to applications running on the local host.
1	Forwards the multicast datagram to hosts on the local subnet.
2 - 255	With a multicast router attached to the sending host's network, forwards multicast datagrams beyond the local subnet. Multicast routers forward the datagram to known networks that have hosts belonging to the specified multicast group. The TTL value is decremented by each multicast router in the path. When the TTL value is decremented to zero, the datagram is no longer forwarded.

10

Multicast Send Program

```
#include <stdio.h>      /* for fprintf() */
#include <sys/socket.h> /* for socket(), connect(), send(), and recv() */
#include <arpa/inet.h>  /* for sockaddr_in and inet_addr() */
#include <stdlib.h>     /* for atoi() and exit() */
#include <string.h>     /* for memset() */
#include <unistd.h>     /* for sleep() */
void ExitWithError(char *errorMessage);  /* External error handling function */

int main(int argc, char *argv[])
{
    int sock;                        /* Socket */
    struct sockaddr_in multicastAddr; /* Multicast address */
    char *multicastIP;               /* IP Multicast address */
    unsigned short multicastPort;    /* Server port */
    char *sendString;                /* String to multicast */
    unsigned char multicastTTL;      /* TTL of multicast packets */
    unsigned int sendStringLen;      /* Length of string to multicast */

    if ((argc < 4) || (argc > 5))    /* Test for correct number of parameters */
    {
        fprintf(stderr,"Usage:  %s <Multicast Address> <Port> <Send String> [<TTL>]\n",
                argv[0]);
        exit(1);
    }

    multicastIP = argv[1];           /* First arg:  multicast IP address */
    multicastPort = atoi(argv[2]);   /* Second arg:  multicast port */
    sendString = argv[3];            /* Third arg:  String to multicast */

    if (argc == 5)                   /* Is TTL specified on command-line? */
        multicastTTL = atoi(argv[4]); /* Command-line specified TTL */
    else
        multicastTTL = 1;            /* Default TTL = 1 */
```

11

Multicast Send Program (Cont.)

```
    /* Create socket for sending/receiving datagrams */
    if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
        ExitWithError("socket() failed");

    /* Set TTL of multicast packet */
    if (setsockopt(sock, IPPROTO_IP, IP_MULTICAST_TTL, (void *) &multicastTTL,
            sizeof(multicastTTL)) < 0)
        ExitWithError("setsockopt() failed");

    /* Construct local address structure */
    memset(&multicastAddr, 0, sizeof(multicastAddr));   /* Zero out structure */
    multicastAddr.sin_family = AF_INET;             /* Internet address family */
    multicastAddr.sin_addr.s_addr = inet_addr(multicastIP);/* Multicast IP address */
    multicastAddr.sin_port = htons(multicastPort);      /* Multicast port */

    sendStringLen = strlen(sendString);  /* Find length of sendString */
    for (;;) /* Run forever */
    {
        /* Multicast sendString in datagram to clients every 3 seconds */
        if (sendto(sock, sendString, sendStringLen, 0, (struct sockaddr *)
                &multicastAddr, sizeof(multicastAddr)) != sendStringLen)
            ExitWithError("sendto() sent a different number of bytes than expected");
        sleep(3);
    }
    /* NOT REACHED */
}

void ExitWithError(char *errorMessage)
{
    perror(errorMessage);
    exit(1);
}
```

12

Multicast Receive Program

```
#include <stdio.h>      /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket(), connect(), sendto(), and recvfrom() */
#include <arpa/inet.h>  /* for sockaddr_in and inet_addr() */
#include <stdlib.h>     /* for atoi() and exit() */
#include <string.h>     /* for memset() */
#include <unistd.h>     /* for close() */
#define MAXRECVSTRING 255  /* Longest string to receive */

void ExitWithError(char *errorMessage);  /* External error handling function */

int main(int argc, char *argv[])
{
    int sock;                        /* Socket */
    struct sockaddr_in multicastAddr; /* Multicast Address */
    char *multicastIP;               /* IP Multicast Address */
    unsigned short multicastPort;    /* Port */
    char recvString[MAXRECVSTRING+1]; /* Buffer for received string */
    int recvStringLen;               /* Length of received string */
    struct ip_mreq multicastRequest;  /* Multicast address join structure */

    if (argc != 3)    /* Test for correct number of arguments */
    {
        fprintf(stderr,"Usage: %s <Multicast IP> <Multicast Port>\n", argv[0]);
        exit(1);
    }

    multicastIP = argv[1];       /* First arg: Multicast IP address (dotted quad) */
    multicastPort = atoi(argv[2]);/* Second arg: Multicast port */

    /* Create a best-effort datagram socket using UDP */
    if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
        ExitWithError("socket() failed");
```


Multicast Receive Program (Cont.)

```
/* Construct bind structure */
    memset(&multicastAddr, 0, sizeof(multicastAddr));   /* Zero out structure */
    multicastAddr.sin_family = AF_INET;                 /* Internet address family */
    multicastAddr.sin_addr.s_addr = htonl(INADDR_ANY);  /* Any incoming interface */
    multicastAddr.sin_port = htons(multicastPort);      /* Multicast port */

    /* Bind to the multicast port */
    if (bind(sock, (struct sockaddr *) &multicastAddr, sizeof(multicastAddr)) < 0)
        ExitWithError("bind() failed");

    /* Specify the multicast group */
    multicastRequest.imr_multiaddr.s_addr = inet_addr(multicastIP);
    /* Accept multicast from any interface */
    multicastRequest.imr_interface.s_addr = htonl(INADDR_ANY);
    /* Join the multicast address */
    if (setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP, (void *) &multicastRequest,
            sizeof(multicastRequest)) < 0)
        ExitWithError("setsockopt() failed");

    /* Receive a single datagram from the server */
    if ((recvStringLen = recvfrom(sock, recvString, MAXRECVSTRING, 0, NULL, 0)) < 0)
        ExitWithError("recvfrom() failed");
    recvString[recvStringLen] = '\0';
    printf("Received: %s\n", recvString);    /* Print the received string */
    close(sock);
    exit(0);
}

void ExitWithError(char *errorMessage)
{
    perror(errorMessage);
    exit(1);
}
```


More Broadcast/Multicast Examples

- Some broadcast sample programs can be found in “unpv13e/bcast” directory.
- Some multicast sample programs can be found in “unpv13e/mcast” directory.