<u>Multicast SENDER</u>

```
#include <sys/types.h>   /* for type definitions */
#include <sys/socket.h>  /* for socket API function calls */
#include <netinet/in.h>  /* for address structs */
#include <arpa/inet.h>   /* for sockaddr_in */
#include <stdio.h>       /* for printf() */
#include <stdlib.h>      /* for atoi() */
#include <string.h>      /* for strlen() */
#include <unistd.h>      /* for close() */

#define MAX_LEN  1024    /* maximum string size to send
*/
#define MIN_PORT 1024    /* minimum port allowed */
#define MAX_PORT 65535   /* maximum port allowed */

int main(int argc, char *argv[]) {

  int sock;                   /* socket descriptor */
  char send_str[MAX_LEN];     /* string to send */
  struct sockaddr_in mc_addr; /* socket address structure
*/
  unsigned int send_len;      /* length of string to send */
  char* mc_addr_str;          /* multicast IP address */
  unsigned short mc_port;     /* multicast port */
  unsigned char mc_ttl=1;     /* time to live (hop count) */

  /* validate number of arguments */
  if (argc != 3) {
    fprintf(stderr,
            "Usage: %s <Multicast IP> <Multicast Port>\n",
            argv[0]);
    exit(1);
  }
  mc_addr_str = argv[1];      /* arg 1: multicast IP address
*/
  mc_port     = atoi(argv[2]); /* arg 2: multicast port
number */

  /* validate the port range */
  if ((mc_port < MIN_PORT) || (mc_port > MAX_PORT)) {
    fprintf(stderr, "Invalid port number argument %d.\n",
            mc_port);
    fprintf(stderr, "Valid range is between %d and %d.\n",
            MIN_PORT, MAX_PORT);
    exit(1);
  }

  /* create a socket for sending to the multicast address */
  if ((sock = socket(PF_INET, SOCK_DGRAM,
IPPROTO_UDP)) < 0) {
    perror("socket() failed");
    exit(1);
  }

  /* set the TTL (time to live/hop count) for the send */
  if ((setsockopt(sock, IPPROTO_IP, IP_MULTICAST_TTL,
       (void*) &mc_ttl, sizeof(mc_ttl))) < 0) {
    perror("setsockopt() failed");
    exit(1);
  }

  /* construct a multicast address structure */
  memset(&mc_addr, 0, sizeof(mc_addr));
  mc_addr.sin_family      = AF_INET;
  mc_addr.sin_addr.s_addr = inet_addr(mc_addr_str);
  mc_addr.sin_port        = htons(mc_port);
  printf("Begin typing (return to send, ctrl-C to quit):\n");

  /* clear send buffer */
  memset(send_str, 0, sizeof(send_str));

  while (fgets(send_str, MAX_LEN, stdin)) {
    send_len = strlen(send_str);

    /* send string to multicast address */
    if ((sendto(sock, send_str, send_len, 0,
         (struct sockaddr *) &mc_addr,
         sizeof(mc_addr))) != send_len) {
      perror("sendto() sent incorrect number of bytes");
      exit(1);
    }

    /* clear send buffer */
    memset(send_str, 0, sizeof(send_str));
  }

  close(sock);

  exit(0);
}
```

Multicast RECEIVER

```
#include <sys/types.h>  /* for type definitions */
#include <sys/socket.h> /* for socket API calls */
#include <netinet/in.h> /* for address structs */
#include <arpa/inet.h>  /* for sockaddr_in */
#include <stdio.h>      /* for printf() and fprintf() */
#include <stdlib.h>     /* for atoi() */
#include <string.h>     /* for strlen() */
#include <unistd.h>     /* for close() */

#define MAX_LEN  1024   /* maximum receive string size
*/
#define MIN_PORT 1024   /* minimum port allowed */
#define MAX_PORT 65535  /* maximum port allowed */

int main(int argc, char *argv[]) {
  int sock;                     /* socket descriptor */
  int flag_on = 1;              /* socket option flag */
  struct sockaddr_in mc_addr;   /* socket address structure
*/
  char recv_str[MAX_LEN+1];     /* buffer to receive string
*/
  int recv_len;                 /* length of string received */
  struct ip_mreq mc_req;        /* multicast request
structure */
  char* mc_addr_str;            /* multicast IP address */
  unsigned short mc_port;       /* multicast port */
  struct sockaddr_in from_addr; /* packet source */
  unsigned int from_len;        /* source addr length */

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

  mc_addr_str = argv[1];      /* arg 1: multicast ip address
*/
  mc_port = atoi(argv[2]);    /* arg 2: multicast port
number */

  /* validate the port range */
  if ((mc_port < MIN_PORT) || (mc_port > MAX_PORT)) {
    fprintf(stderr, "Invalid port number argument %d.\n",
          mc_port);
    fprintf(stderr, "Valid range is between %d and %d.\n",
          MIN_PORT, MAX_PORT);
    exit(1);
  }

  /* create socket to join multicast group on */
  if ((sock = socket(PF_INET, SOCK_DGRAM,
IPPROTO_UDP)) < 0) {
    perror("socket() failed");
    exit(1);
  }

  /* set reuse port to on to allow multiple binds per host */
  if ((setsockopt(sock, SOL_SOCKET, SO_REUSEADDR,
&flag_on,
       sizeof(flag_on))) < 0) {
    perror("setsockopt() failed");
    exit(1);
  }
```

```
/* construct a multicast address structure */
memset(&mc_addr, 0, sizeof(mc_addr));
mc_addr.sin_family      = AF_INET;
mc_addr.sin_addr.s_addr = htonl(INADDR_ANY);
mc_addr.sin_port        = htons(mc_port);

/* bind to multicast address to socket */
if ((bind(sock, (struct sockaddr *) &mc_addr,
     sizeof(mc_addr))) < 0) {
  perror("bind() failed");
  exit(1);
}

/* construct an IGMP join request structure */
mc_req.imr_multiaddr.s_addr = inet_addr(mc_addr_str);
mc_req.imr_interface.s_addr = htonl(INADDR_ANY);

/* send an ADD MEMBERSHIP message via setsockopt */
if ((setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP,
     (void*) &mc_req, sizeof(mc_req))) < 0) {
  perror("setsockopt() failed");
  exit(1);
}

for (;;) {          /* loop forever */
  /* clear the receive buffers & structs */
  memset(recv_str, 0, sizeof(recv_str));
  from_len = sizeof(from_addr);
  memset(&from_addr, 0, from_len);

  /* block waiting to receive a packet */
  if ((recv_len = recvfrom(sock, recv_str, MAX_LEN, 0,
       (struct sockaddr*)&from_addr, &from_len)) < 0) {
    perror("recvfrom() failed");
```

```
      exit(1);
    }

    /* output received string */
    printf("Received %d bytes from %s: ", recv_len,
           inet_ntoa(from_addr.sin_addr));
    printf("%s", recv_str);
  }

  /* send a DROP MEMBERSHIP message via setsockopt */
  if ((setsockopt(sock, IPPROTO_IP, IP_DROP_MEMBERSHIP,
       (void*) &mc_req, sizeof(mc_req))) < 0) {
    perror("setsockopt() failed");
    exit(1);
  }

  close(sock);
}
```