```
/*
 *  dataServer.c
 *
 *  Example of a server that receives messages from data-requesting clients
 *      and pulses from one or more data-supplying threads.
 *
*/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/neutrino.h>
#include <pthread.h>

// message send definitions

// messages
#define MT_WAIT_DATA        2       // message from client

// pulses
#define CODE_SEND_DATA      3       // pulse with data

// message reply definitions
#define MT_OK               0       // message to client

// message structure
typedef struct
{
    int messageType;                // contains both message to and from client
    int messageData;                // optional data, depending upon message
} ClientMessageT;

typedef union
{
    ClientMessageT  msg;            // a message can be either from a client, or
    struct _pulse   pulse;          // a pulse
} MessageT;

// client table
#define MAX_CLIENT 16               // maximum number of simultaneous clients

struct
{
    int in_use;                     // is this client entry in use?
    int rcvid;                      // receive ID of client
}   clients [MAX_CLIENT];           // client table

int     chid;                       // channel ID (global)
int     debug = 1;                  // set debug value, 1 == enabled, 0 == off
char    *progname = "dataServer.c";

// forward prototypes
static  void *receiveMessages (void *not_used);
static  void gotAPulse (struct _pulse *pulse);
static  void gotAMessage (int rcvid, ClientMessageT *msg);
```

```
int
main (void)                         // ignore command-line arguments
{

    if ((chid = ChannelCreate (0)) == -1) {
        fprintf (stderr, "%s:  couldn't create channel!\n", progname);
        perror (NULL);
        exit (EXIT_FAILURE);
    }

//  pthread_create (NULL, NULL, receiveMessages, NULL);
    receiveMessages(NULL);

    // you'll never get here
    return (EXIT_SUCCESS);
}

void *
receiveMessages (void *not_used)
{
    int rcvid;                      // process ID of the sender
    MessageT msg;                   // the message itself

    // receive messages
    for (;;) {
        rcvid = MsgReceive (chid, &msg, sizeof (msg), NULL);

        // determine who the message came from
        if (rcvid == 0) {
            // production code should check "code" field...
            gotAPulse (&msg.pulse);
        } else {
            gotAMessage (rcvid, &msg.msg);
        }
    };

    return NULL;
}
```

```
/*
 *  gotAPulse
 *
 *
 *  This routine handles a pulse meaning "here's some
 *  data".  It runs through the list of clients to see
 *  which client wants the data, and replies to the client with the data.
 *  For simplicity, we'll assume
 *  that data doesn't wait around if nobody immediately wants it.
 *
 */

void
gotAPulse (struct _pulse *pulse)
{
    ClientMessageT  msg;
    int             i;

    if (debug) {
        time_t  now;

        time (&now);
        printf ("Got a Pulse at %s", ctime (&now));
    }
    // see if we can find a client to reply to with
    //  data from the received pulse.
    for (i = 0; i < MAX_CLIENT; i++) {

        if (clients [i].in_use) {

            // found one
            msg.messageType = MT_OK;
            msg.messageData = pulse->value.sival_int;

            // reply to the waiting CLIENT!
            MsgReply (clients [i].rcvid, EOK, &msg, sizeof (msg));

            clients [i].in_use = 0;
            return;
        }
    }

    fprintf (stderr, "Table empty, pulse ignored\n");
    return;
}
```

```c
/*
 *  gotAMessage
 *
 *  This routine is called whenever a message arrives.  The
 *  type of message will always be a "wait for data" message
 *  in this version of the code, and so the routine will act accordingly.
*/

void
gotAMessage (int rcvid, ClientMessageT *msg)
{
    int i;

    // determine the kind of message that it is
    switch (msg -> messageType) {

    // client wants to wait for data
    case    MT_WAIT_DATA:

        // see if we can find a blank spot in the client table
        for (i = 0; i < MAX_CLIENT; i++) {

            if (!clients [i].in_use) {

                // found one -- mark as in use, save rcvid
                clients [i].in_use = 1;
                clients [i].rcvid = rcvid;
                return;
            }
        }

        fprintf (stderr, "Table full, message from rcvid %d ignored, "
                         "client blocked\n", rcvid);
        break;
    }
}
```