# prog3.c

```
#include <stdio.h>
#define LINKCHANGES 1
/* ******************************************************************
Programming assignment 3: implementing distributed, asynchronous,
distance vector routing.
THIS IS THE MAIN ROUTINE.  IT SHOULD NOT BE TOUCHED AT ALL BY STUDENTS!
********************************************************************/
/* a rtpkt is the packet sent from one routing update process to
another via the call tolayer3() */
struct rtpkt {
     int sourceid;      /* id of sending router sending this pkt */
     int destid;        /* id of router to which pkt being sent
                             (must be an immediate neighbor) */
     int mincost[4];    /* min cost to node 0 ... 3 */
};

int TRACE = 1;            /* for my debugging */
int YES = 1;
int NO = 0;

/* external function prototypes - needed for the compiler */
extern void rtinit0();
extern void rtinit1();
extern void rtinit2();
extern void rtinit3();
extern void rtupdate0(struct rtpkt *rcvdpkt);
extern void rtupdate1(struct rtpkt *rcvdpkt);
extern void rtupdate2(struct rtpkt *rcvdpkt);
extern void rtupdate3(struct rtpkt *rcvdpkt);
extern void linkhandler0(int linkid, int newcost);
extern void linkhandler1(int linkid, int newcost);

/* local function prototypes - needed for the compiler */
void init();
void insertevent(struct event *p);
void printevlist();

creatertpkt(initrtpkt, srcid, destid, mincosts)
struct rtpkt *initrtpkt;
int srcid;
int destid;
int mincosts[];
{
     int i;
     initrtpkt->sourceid = srcid;
     initrtpkt->destid = destid;
     for (i = 0; i<4; i++)
          initrtpkt->mincost[i] = mincosts[i];
}


/******************************************************************
```

```
***************** NETWORK EMULATION CODE STARTS BELOW ***********
The code below emulates the layer 2 and below network environment:
- emulates the tranmission and delivery (with no loss and no
corruption) between two physically connected nodes
- calls the initializations routines rtinit0, etc., once before
beginning emulation

THERE IS NOT REASON THAT ANY STUDENT SHOULD HAVE TO READ OR UNDERSTAND
THE CODE BELOW.  YOU SHOLD NOT TOUCH, OR REFERENCE (in your code) ANY
OF THE DATA STRUCTURES BELOW.  If you're interested in how I designed
the emulator, you're welcome to look at the code - but again, you should have
to, and you defeinitely should not have to modify
******************************************************************/
struct event {
     float evtime;                    /* event time */
     int evtype;                      /* event type code */
     int eventity;                    /* entity where event occurs */
     struct rtpkt *rtpktptr;          /* ptr to packet (if any) assoc w/ this event */
     struct event *prev;
     struct event *next;
};
struct event *evlist = NULL;          /* the event list */
                                      /* possible events: */
#define  FROM_LAYER2     2
#define  LINK_CHANGE     10

float clocktime = 0.000;

main()
{
     struct event *eventptr;

     init();


     while (1) {

          eventptr = evlist;           /* get next event to simulate */
          if (eventptr == NULL)
               goto terminate;
          evlist = evlist->next;       /* remove this event from event list */
          if (evlist != NULL)
               evlist->prev = NULL;
          if (TRACE>1) {
               printf("MAIN: rcv event, t=%.3f, at %d",
                    eventptr->evtime, eventptr->eventity);
               if (eventptr->evtype == FROM_LAYER2) {
                    printf(" src:%2d,", eventptr->rtpktptr->sourceid);
                    printf(" dest:%2d,", eventptr->rtpktptr->destid);
                    printf(" contents: %3d %3d %3d %3d\n",
                         eventptr->rtpktptr->mincost[0], eventptr->rtpktptr->mincost[1],
                         eventptr->rtpktptr->mincost[2], eventptr->rtpktptr->mincost[3]);
               }
          }
          clocktime = eventptr->evtime;    /* update time to next event time */
```

```
        if (eventptr->evtype == FROM_LAYER2) {
            if (eventptr->eventity == 0)
                rtupdate0(eventptr->rtpktptr);
            else if (eventptr->eventity == 1)
                rtupdate1(eventptr->rtpktptr);
            else if (eventptr->eventity == 2)
                rtupdate2(eventptr->rtpktptr);
            else if (eventptr->eventity == 3)
                rtupdate3(eventptr->rtpktptr);
            else { printf("Panic: unknown event entity\n"); exit(0); }
        }
        else if (eventptr->evtype == LINK_CHANGE) {
            if (clocktime<10001.0) {
                linkhandler0(1, 20);
                linkhandler1(0, 20);
            }
            else {                  linkhandler0(1, 1);
                linkhandler1(0, 1);
            }
        }
        else
        {
            printf("Panic: unknown event type\n"); exit(0);
        }
        if (eventptr->evtype == FROM_LAYER2)
            free(eventptr->rtpktptr);       /* free memory for packet, if any */
        free(eventptr);                  /* free memory for event struct   */
    }

terminate:
    printf("\nSimulator terminated at t=%f, no packets in medium\n", clocktime);
}

void init()                       /* initialize the simulator */
{
    int i;
    float sum, avg;
    float jimsrand();
    struct event *evptr;

    printf("Enter TRACE:");
    scanf("%d", &TRACE);

    srand(9999);             /* init random number generator */
    sum = 0.0;               /* test random number generator for students */
    for (i = 0; i<1000; i++)
        sum = sum + jimsrand();    /* jimsrand() should be uniform in [0,1] */
    avg = sum / 1000.0;
    avg = 0.5;               /* visual studio */
    if (avg < 0.25 || avg > 0.75) {
        printf("It is likely that random number generation on your machine\n");
        printf("is different from what this emulator expects.  Please take\n");
        printf("a look at the routine jimsrand() in the emulator code. Sorry. \n");
        exit(0);
    }
```

```
     clocktime = 0.0;              /* initialize time to 0.0 */
     rtinit0();
     rtinit1();
     rtinit2();
     rtinit3();

     /* initialize future link changes */
     if (LINKCHANGES == 1) {
          evptr = (struct event *)malloc(sizeof(struct event));
          evptr->evtime = 10000.0;
          evptr->evtype = LINK_CHANGE;
          evptr->eventity = -1;
          evptr->rtpktptr = NULL;
          insertevent(evptr);
          evptr = (struct event *)malloc(sizeof(struct event));
          evptr->evtype = LINK_CHANGE;
          evptr->evtime = 20000.0;
          evptr->rtpktptr = NULL;
          insertevent(evptr);
     }
}

/****************************************************************************/
/* jimsrand(): return a float in range [0,1].  The routine below is used to */
/* isolate all random number generation in one location.  We assume that the*/
/* system-supplied rand() function return an int in therange [0,mmm]       */
/****************************************************************************/
float jimsrand()
{
     double mmm = 2147483647;   /* largest int  - MACHINE DEPENDENT!!!!!!!!   */
     float x;                 /* individual students may need to change mmm */
     x = rand() / mmm;           /* x should be uniform in [0,1] */
     return(x);
}

/******************** EVENT HANDLINE ROUTINES *******/
/*  The next set of routines handle the event list   */
/*****************************************************/

void insertevent(p)
struct event *p;
{
     struct event *q, *qold;

     if (TRACE>3) {
          printf("           INSERTEVENT: time is %lf\n", clocktime);
          printf("           INSERTEVENT: future time will be %lf\n", p->evtime);
     }
     q = evlist;     /* q points to header of list in which p struct inserted */
     if (q == NULL) {   /* list is empty */
          evlist = p;
          p->next = NULL;
          p->prev = NULL;
     }
     else {
          for (qold = q; q != NULL && p->evtime > q->evtime; q = q->next)
               qold = q;
          if (q == NULL) {   /* end of list */
               qold->next = p;
               p->prev = qold;
               p->next = NULL;
          }
          else if (q == evlist) { /* front of list */
               p->next = evlist;
               p->prev = NULL;
               p->next->prev = p;
               evlist = p;
          }
          else {     /* middle of list */
               p->next = q;
               p->prev = q->prev;
               q->prev->next = p;
               q->prev = p;
          }
     }
}

void printevlist()
{
     struct event *q;
     printf("--------------\nEvent List Follows:\n");
     for (q = evlist; q != NULL; q = q->next) {
          printf("Event time: %f, type: %d entity: %d\n", q->evtime, q->evtype, q->eventity);
     }
     printf("--------------\n");
}

/************************* TOLAYER2 ***************/
void tolayer2(packet)
struct rtpkt packet;

{
     struct rtpkt *mypktptr;
     struct event *evptr, *q;
     float jimsrand(), lastime;
     int i;

     int connectcosts[4][4];

     /* initialize by hand since not all compilers allow array initilization */
     connectcosts[0][0] = 0;  connectcosts[0][1] = 1;  connectcosts[0][2] = 3;
     connectcosts[0][3] = 7;
     connectcosts[1][0] = 1;  connectcosts[1][1] = 0;  connectcosts[1][2] = 1;
     connectcosts[1][3] = 999;
     connectcosts[2][0] = 3;  connectcosts[2][1] = 1;  connectcosts[2][2] = 0;
     connectcosts[2][3] = 2;
     connectcosts[3][0] = 7;  connectcosts[3][1] = 999;  connectcosts[3][2] = 2;
     connectcosts[3][3] = 0;
```

```
    /* be nice: check if source and destination id's are reasonable */
    if (packet.sourceid<0 || packet.sourceid >3) {
        printf("WARNING: illegal source id in your packet, ignoring packet!\n");
        return;
    }
    if (packet.destid<0 || packet.destid >3) {
        printf("WARNING: illegal dest id in your packet, ignoring packet!\n");
        return;
    }
    if (packet.sourceid == packet.destid) {
        printf("WARNING: source and destination id's the same, ignoring packet!\n");
        return;
    }
    if (connectcosts[packet.sourceid][packet.destid] == 999) {
        printf("WARNING: source and destination not connected, ignoring packet!\n");
        return;
    }

    /* make a copy of the packet student just gave me since he/she may decide */
    /* to do something with the packet after we return back to him/her */
    mypktptr = (struct rtpkt *) malloc(sizeof(struct rtpkt));
    mypktptr->sourceid = packet.sourceid;
    mypktptr->destid = packet.destid;
    for (i = 0; i<4; i++)
        mypktptr->mincost[i] = packet.mincost[i];
    if (TRACE>2) {
        printf("    TOLAYER2: source: %d, dest: %d\n             costs:",
            mypktptr->sourceid, mypktptr->destid);
        for (i = 0; i<4; i++)
            printf("%d  ", mypktptr->mincost[i]);
        printf("\n");
    }

    /* create future event for arrival of packet at the other side */
    evptr = (struct event *)malloc(sizeof(struct event));
    evptr->evtype = FROM_LAYER2;   /* packet will pop out from layer3 */
    evptr->eventity = packet.destid; /* event occurs at other entity */
    evptr->rtpktptr = mypktptr;      /* save ptr to my copy of packet */

                                        /* finally, compute the arrival time of packet at the other end.
                                        medium can not reorder, so make sure packet arrives between 1 and
10

                                        time units after the latest arrival time of packets
                                        currently in the medium on their way to the destination */
    lastime = clocktime;
    for (q = evlist; q != NULL; q = q->next)
        if ((q->evtype == FROM_LAYER2  && q->eventity == evptr->eventity))
            lastime = q->evtime;
    evptr->evtime = lastime + 2.*jimsrand();


    if (TRACE>2)
        printf("    TOLAYER2: scheduling arrival on other side\n");
    insertevent(evptr);
}
```