#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define BIDIRECTIONAL 0

/\*\*\*\*\*\*\*\*\* SECTION 0: GLOBAL DATA STRUCTURES\*\*\*\*\*\*\*\*\*/

/\* a "msg" is the data unit passed from layer 5 (teachers code) to layer \*/

/\* 4 (students' code). It contains the data (characters) to be delivered \*/

/\* to layer 5 via the students transport level protocol entities. \*/

struct msg {

char data[20];

};

/\* a packet is the data unit passed from layer 4 (students code) to layer \*/

/\* 3 (teachers code). Note the pre-defined packet structure, which all \*/

/\* students must follow. \*/

struct pkt {

int seqnum;

int acknum;

int checksum;

char payload[20];

};

void ComputeChecksum(struct pkt \*);

int CheckCorrupted(struct pkt);

void A\_output(struct msg);

void tolayer3(int,struct pkt);

int starttimer(int, float);

void A\_input(struct pkt);

void B\_input(struct pkt);

void B\_init(void);

void A\_init(void);

void tolayer5(int, char\*);

void tolayer3(int, struct pkt);

int stoptimer(int);

float currenttime(void);

void printevlist(void);

void generate\_next\_arrival(void);

void init(void);

void A\_timerinterrupt(void);

/\*\*\*\*\*\*\*\*\* SECTION I: GLOBAL VARIABLES\*\*\*\*\*\*\*\*\*/

/\* the following global variables will be used by the routines to be implemented.

you may define new global variables if necessary. However, you should reduce

the number of new global variables to the minimum. Excessive new global variables

will result in point deduction.

\*/

#define MAXBUFSIZE 5000

#define RTT 15.0

#define NOTUSED 0

#define TRUE 1

#define FALSE 0

#define A 0

#define B 1

int WINDOWSIZE=8;

int expectedseqnum; /\* expected sequence number at receiver side \*/

float currenttime(); /\* get the current time \*/

int nextseqnum; /\* next sequence number to use in sender side \*/

int base; /\* the head of sender window \*/

struct pkt\* winbuf; /\* window packets buffer \*/

int winfront,winrear; /\* front and rear points of window buffer \*/

int pktnum; /\* the # of packets in window buffer, i.e., the # of packets to be resent when timeout \*/

struct msg buffer[MAXBUFSIZE]; /\* sender message buffer \*/

int buffront,bufrear; /\* front and rear pointers of buffer \*/

int msgnum; /\* # of messages in buffer \*/

int totalmsg=-1;

int packet\_lost =0;

int packet\_corrupt=0;

int packet\_sent =0;

int packet\_correct=0;

int packet\_resent =0;

int packet\_timeout=0;

/\*\*\*\*\*\*\*\*\* SECTION II: FUNCTIONS TO BE COMPLETED BY STUDENTS\*\*\*\*\*\*\*\*\*/

/\* layer 5: application layer which calls functions of layer 4 to send messages; \*/

/\* layer 4: transport layer (where your program is implemented); \*/

/\* layer 3: networking layer which calls functions of layer 4 to deliver the messages that have arrived. \*/

/\* compute the checksum of the packet to be sent from the sender \*/

void ComputeChecksum(packet)

struct pkt \*packet;

{

}

/\* check the checksum of the packet received, return TRUE if packet is corrupted, FALSE otherwise \*/

int CheckCorrupted(packet)

struct pkt packet;

{

return 0;

}

/\* called from layer 5, passed the data to be sent to other side \*/

void A\_output(message)

struct msg message;

{

}

/\* called from layer 3, when a packet arrives for layer 4 \*/

void A\_input(packet)

struct pkt packet;

{

};

/\* called when A's timer goes off \*/

void A\_timerinterrupt()

{

}

/\* called from layer 3, when a packet arrives for layer 4 at B\*/

void B\_input(packet)

struct pkt packet;

{

}

/\* the following routine will be called once (only) before any other \*/

/\* entity A routines are called. You can use it to do any initialization \*/

void A\_init()

{

base = 0;

nextseqnum = 0;

buffront = 0;

bufrear = 0;

msgnum = 0;

winfront = 0;

winrear = 0;

pktnum = 0;

}

/\* the following rouytine will be called once (only) before any other \*/

/\* entity B routines are called. You can use it to do any initialization \*/

void B\_init()

{

expectedseqnum = 0;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SECTION III: NETWORK EMULATION CODE \*\*\*\*\*\*\*\*\*\*\*

The code below emulates the layer 3 and below network environment:

- emulates the tranmission and delivery (possibly with bit-level corruption

and packet loss) of packets across the layer 3/4 interface

- handles the starting/stopping of a timer, and generates timer

interrupts (resulting in calling students timer handler).

- generates message to be sent (passed from later 5 to 4)

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 defeinitely should not have to modify

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct event {

float evtime; /\* event time \*/

int evtype; /\* event type code \*/

int eventity; /\* entity where event occurs \*/

struct pkt \*pktptr; /\* ptr to packet (if any) assoc w/ this event \*/

struct event \*prev;

struct event \*next;

};

struct event \*evlist = NULL; /\* the event list \*/

void insertevent(struct event \*);

/\* possible events: \*/

#define TIMER\_INTERRUPT 0

#define FROM\_LAYER5 1

#define FROM\_LAYER3 2

#define OFF 0

#define ON 1

#define A 0

#define B 1

int TRACE = -1; /\* for my debugging \*/

int nsim = 0; /\* number of messages from 5 to 4 so far \*/

int nsimmax = 0; /\* number of msgs to generate, then stop \*/

float time = 0.000;

float lossprob; /\* probability that a packet is dropped \*/

float corruptprob; /\* probability that one bit is packet is flipped \*/

float lambda; /\* arrival rate of messages from layer 5 \*/

int ntolayer3; /\* number sent into layer 3 \*/

int nlost; /\* number lost in media \*/

int ncorrupt; /\* number corrupted by media\*/

char pattern[40]; /\*channel pattern string\*/

int npttns=0;

int cp=-1; /\*current pattern\*/

char pttnchars[3]={'o','-','x'};

enum pttns {OK=0,CORRUPT,LOST};

void main(void)

{

struct event \*eventptr;

struct msg msg2give;

struct pkt pkt2give;

int i,j;

init();

A\_init();

B\_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>=2) {

printf("\nEVENT time: %f,",eventptr->evtime);

printf(" type: %d",eventptr->evtype);

if (eventptr->evtype==0)

printf(", timerinterrupt ");

else if (eventptr->evtype==1)

printf(", fromlayer5 ");

else

printf(", fromlayer3 ");

printf(" entity: %d\n",eventptr->eventity);

}

time = eventptr->evtime; /\* update time to next event time \*/

//if (nsim==nsimmax)

// break; /\* all done with simulation \*/

if (eventptr->evtype == FROM\_LAYER5 ) {

generate\_next\_arrival(); /\* set up future arrival \*/

/\* fill in msg to give with string of same letter \*/

j = nsim % 26;

for (i=0; i<20; i++)

msg2give.data[i] = 97 + j;

if (TRACE>2) {

printf(" MAINLOOP: data given to student: ");

for (i=0; i<20; i++)

printf("%c", msg2give.data[i]);

printf("\n");

}

//nsim++;

if (eventptr->eventity == A)

A\_output(msg2give);

else{}

//B\_output(msg2give);

}

else if (eventptr->evtype == FROM\_LAYER3) {

pkt2give.seqnum = eventptr->pktptr->seqnum;

pkt2give.acknum = eventptr->pktptr->acknum;

pkt2give.checksum = eventptr->pktptr->checksum;

for (i=0; i<20; i++)

pkt2give.payload[i] = eventptr->pktptr->payload[i];

if (eventptr->eventity ==A) /\* deliver packet by calling \*/

A\_input(pkt2give); /\* appropriate entity \*/

else

B\_input(pkt2give);

free(eventptr->pktptr); /\* free the memory for packet \*/

}

else if (eventptr->evtype == TIMER\_INTERRUPT) {

if (eventptr->eventity == A)

A\_timerinterrupt();

else{}

//B\_timerinterrupt();

}

else {

printf("INTERNAL PANIC: unknown event type \n");

}

free(eventptr);

}

terminate:

printf("Simulator terminated, [%d] msgs sent from layer5\n",nsim);

//printf(" correctly sent pkts: %d \n", packet\_correct);

//printf(" resent pkts: %d \n", packet\_resent);

}

void init() /\* initialize the simulator \*/

{

float jimsrand();

winbuf=(struct pkt\*)malloc(sizeof(struct pkt)\*WINDOWSIZE);

//FILE \*fp;

//fp = fopen ("parameter","r");

//printf("----- Stop and Wait Network Simulator Version 1.1 -------- \n\n");

printf("Enter the number of messages to simulate: \n");

//fscanf(fp,"%d",&nsimmax);

scanf("%d",&nsimmax);

printf("Enter time between messages from sender's layer5 [ > 0.0]:\n");

//fscanf(fp,"%f",&lambda);

scanf("%f",&lambda);

printf("Enter channel pattern string\n");

//fscanf(fp, "%s",pattern);

scanf("%s",pattern);

npttns=strlen(pattern);

//printf("%d patterns: %s\n",npttns,pattern);

printf("Enter sender's window size\n");

scanf("%d",&WINDOWSIZE);

//printf("Enter TRACE:\n");

//fscanf(fp,"%d",&TRACE);

//scanf("%d",&TRACE);

ntolayer3 = 0;

nlost = 0;

ncorrupt = 0;

time=0.0; /\* initialize time to 0.0 \*/

generate\_next\_arrival(); /\* initialize event list \*/

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* EVENT HANDLINE ROUTINES \*\*\*\*\*\*\*/

/\* The next set of routines handle the event list \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void generate\_next\_arrival()

{

double x,log(),ceil();

struct event \*evptr;

//char \*malloc();

if(nsim>=nsimmax) return;

if (TRACE>2)

printf(" GENERATE NEXT ARRIVAL: creating new arrival\n");

x = lambda;

evptr = (struct event \*)malloc(sizeof(struct event));

evptr->evtime = time + x;

evptr->evtype = FROM\_LAYER5;

if (BIDIRECTIONAL)

evptr->eventity = B;

else

evptr->eventity = A;

insertevent(evptr);

nsim++;

}

void insertevent(p)

struct event \*p;

{

struct event \*q,\*qold;

if (TRACE>2) {

printf(" INSERTEVENT: time is %lf\n",time);

printf(" INSERTEVENT: future time will be %lf\n",p->evtime);

}

q = evlist; /\* q points to front 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");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SECTION IV: Student-callable ROUTINES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* get the current time of the system\*/

float currenttime()

{

return time;

}

/\* called by students routine to cancel a previously-started timer \*/

int stoptimer(AorB)

int AorB; /\* A or B is trying to stop timer \*/

{

struct event \*q;

if (TRACE>2)

printf(" STOP TIMER: stopping timer at %f\n",time);

/\* for (q=evlist; q!=NULL && q->next!=NULL; q = q->next) \*/

for (q=evlist; q!=NULL ; q = q->next)

if ( (q->evtype==TIMER\_INTERRUPT && q->eventity==AorB) ) {

/\* remove this event \*/

if (q->next==NULL && q->prev==NULL)

evlist=NULL; /\* remove first and only event on list \*/

else if (q->next==NULL) /\* end of list - there is one in front \*/

q->prev->next = NULL;

else if (q==evlist) { /\* front of list - there must be event after \*/

q->next->prev=NULL;

evlist = q->next;

}

else { /\* middle of list \*/

q->next->prev = q->prev;

q->prev->next = q->next;

}

free(q);

return 0;

}

printf("Warning: unable to cancel your timer. It wasn't running.\n");

return 0;

};

int starttimer(int AorB,float increment)

{

struct event \*q;

struct event \*evptr;

//char \*malloc();

if (TRACE>2)

printf(" START TIMER: starting timer at %f\n",time);

/\* be nice: check to see if timer is already started, if so, then warn \*/

/\* for (q=evlist; q!=NULL && q->next!=NULL; q = q->next) \*/

for (q=evlist; q!=NULL ; q = q->next)

if ( (q->evtype==TIMER\_INTERRUPT && q->eventity==AorB) ) {

printf("Warning: attempt to start a timer that is already started\n");

return 0;

}

/\* create future event for when timer goes off \*/

evptr = (struct event \*)malloc(sizeof(struct event));

evptr->evtime = time + increment;

evptr->evtype = TIMER\_INTERRUPT;

evptr->eventity = AorB;

insertevent(evptr);

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TOLAYER3 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void tolayer3(AorB,packet)

int AorB; /\* A or B is trying to stop timer \*/

struct pkt packet;

{

struct pkt \*mypktptr;

struct event \*evptr;

//char \*malloc();

float jimsrand();

int i;

cp++;

ntolayer3++;

/\* simulate losses: \*/

if (pattern[cp % npttns] == pttnchars[LOST]){

nlost++;

if (TRACE>0)

printf(" TOLAYER3: packet being lost\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 pkt \*)malloc(sizeof(struct pkt));

mypktptr->seqnum = packet.seqnum;

mypktptr->acknum = packet.acknum;

mypktptr->checksum = packet.checksum;

for (i=0; i<20; i++)

mypktptr->payload[i] = packet.payload[i];

if (TRACE>2) {

printf(" TOLAYER3: seq: %d, ack %d, check: %d ", mypktptr->seqnum,

mypktptr->acknum, mypktptr->checksum);

for (i=0; i<20; i++)

printf("%c",mypktptr->payload[i]);

printf("\n");

}

/\* create future event for arrival of packet at the other side \*/

evptr = (struct event \*)malloc(sizeof(struct event));

evptr->evtype = FROM\_LAYER3; /\* packet will pop out from layer3 \*/

evptr->eventity = (AorB+1) % 2; /\* event occurs at other entity \*/

evptr->pktptr = mypktptr; /\* save ptr to my copy of packet \*/

evptr->evtime = time + RTT/2-1; /\* hard code the delay on channel \*/

/\* simulate corruption: \*/

if (pattern[cp % npttns]== pttnchars[CORRUPT]) {

ncorrupt++;

mypktptr->payload[0]='Z'; /\* corrupt payload \*/

mypktptr->seqnum = 999999;

mypktptr->acknum = 999999;

if (TRACE>0)

printf(" TOLAYER3: packet being corrupted\n");

}

if (TRACE>2)

printf(" TOLAYER3: scheduling arrival on other side\n");

insertevent(evptr);

}

void tolayer5(AorB,datasent)

int AorB;

char datasent[20];

{

int i;

if (TRACE>2) {

printf(" TOLAYER5: data received: ");

for (i=0; i<20; i++)

printf("%c",datasent[i]);

printf("\n");

}

}