CSE 3461/5461 Programming Assignment 3.

1. Goal

- To design the full version of Go-Back-N ARQ protocol.

- Changes from Programming Assignment 2:
  - In the previous assignment, when the info-frame timer expires, it is instructed that all unacknowledged info-frames should be retransmitted. Instead, when the info-frame timer expires, a request (command) RR with appropriate ack_number and P/F_bit=1 should be sent.
  - Then, a receiver of a request RR with P/F_bit=1 should process it normally, i.e. some number of unacknowledged info-frames might be acknowledged and window should be adjusted accordingly. In addition, a response RR with P/F_bit=1 must be sent with appropriate ack_number.
  - Then, when a sender receives a response RR with P/F_bit=1 should process it normally, i.e. some number of unacknowledged info-frames might be acknowledged and window should be adjusted accordingly, and then it should retransmit all unacknowledged frames (if any).

- Thus, this assignment requires that you update your Programming Assignment 2 code according to these go-back-N ARQ requirements.

2. Basic Instructions

- The following sections from Programming Assignment 2 are still valid without any change: 2, 3, 4.2, 4.4

- In the section 4.1. of Programming Assignment 2, only a new integer variable rr_pbit_sent is included in DLC_Conn_Info_TYPE:

```c
typedef struct {
    int snd_nxt, snd_una, rcv_nxt, rej_already_sent, window_size,
    rr_pbit_sent;
    PDU_BUFFER_TYPE pdu_buffer;
} DLC_Conn_Info_TYPE;
```

- rr_pbit_sent: When this variable is 1 indicates that a request RR with P/F_bit=1 has been sent and a response RR with P/F_bit=1 is expected.
3. The Algorithm

The following algorithm describes the events that the datalink layer must process, and the actions that it must take. Parts of algorithm may also be outlined in the skeleton code provided in dlc_layer.c. Note: Although similar, this section has changes from the section 4.3 in Programming Assignment 2.

a. The function FromApplicationToDatalink() of the datalink layer is automatically called when an a_pdu from the application is received:
   • Insert a_pdu into the transmission buffer;
   • Send the info d_pdu to the physical layer, if possible, i.e. if there are less than window_size sent but unacknowledged a_pdu's in the transmission buffer;

b. The function FromPhysicalToDatalink() of the datalink layer is automatically called when a d_pdu from the physical layer is received. If the d_pdu is corrupted, then simply discard it, else:
   • If the d_pdu is a response RR with P/F_bit=0
      o Call UpdatePDUBuffer() to delete a_pdu’s in the transmission buffer that may be now acknowledged.
      o Update snd_una.
      o Send any unsent a_pdu's from the transmission buffer to the physical layer, if there are now less than window_size sent but unacknowledged a_pdu’s in the buffer.
   • If the d_pdu is a request RR with P/F_bit=1
      o Call UpdatePDUBuffer() to delete a_pdu’s in the transmission buffer that may be now acknowledged.
      o Update snd_una and set rej_pbit_sent to 0.
      o Send any unsent a_pdu's from the transmission buffer to the physical layer, if there are now less than window_size sent but unacknowledged a_pdu’s in the buffer.
      o Send a response RR with P/F_bit=1 and appropriate ack number.
   • If the d_pdu is a response RR with P/F_bit=1 and rr_pbit_sent=1
      o Call UpdatePDUBuffer() to delete a_pdu’s in the transmission buffer that may be now acknowledged.
      o Update snd_una and set rr_pbit_sent=0.
      o Now retransmits all unacknowledged a_pdu's. In order to do this, snd_nxt must be set to snd_una and then send as many a_pdu's from the transmission buffer as allowed by the window (similarly as you did in the function DatalinkTimerExpired() of Programming Assignment 2).
   • All other RR d_pdu’s should be discarded.
• If the d_pdu is an response REJ with P/F_bit=0
  o Call UpdatePDUBuffer() to delete a_pdu’s in the transmission buffer that may be now acknowledged.
  o Update snd_una and snd_nxt.
  o Send up to window size a_pdu's from the transmission buffer to the physical layer, if there are still a_pdu’s in the buffer. Note that for some of those a_pdu’s being sent this might be retransmission.

• All other REJ d_pdu’s should be discarded.

• If the d_pdu is a request info frame with P/F_bit=0
  o Check the sequence number of the info d_pdu. If this info frame is out of sequence then discard the d_pdu and if rej_already_sent is 0 then set rej_already_sent to 1 and send one REJ. If rej_already_send is already 1, just discard this info frame (and do not send REJ nor RR).
  o If the d_pdu is the one expected, then increment rcv_nxt, send an RR, reset rej_already_sent and send the a_pdu to application.

• All other info frame d_pdu’s should be dicarded.

• To send an info d_pdu to the physical layer, do the following
  • Get a pointer to an a_pdu from the transmission buffer.
  • Create a d_pdu and copy the contents from the a_pdu to it.
  o Set the remaining fields of the d_pdu; Set P/F_bit=0;
    Remember that info frames are always requests;
  o Start the retransmission timeout;
  o Send the d_pdu to the physical layer.
  o Increment snd_nxt.

• To send a RR or a REJ, create a d_pdu, fill in the needed fields, and send it to the physical layer; Remember that REJ is always response with P/F_bit=0, while RR could be a response with P/F_bit set to 0 or 1 or a request with P/F_bit=1;

• Remember to free every pdu, received either from the application or from the physical layer, after its processing is done.

c. When the retransmission timeout expires, the function DatalinkTimerExpired() is automatically called. In this function, if there are a_pdu in the transmission buffer, send a request RR with P/F_bit=1 and rcv_nxt number, set rr_pbit_sent=1 and start the retransmission timer.
4. Methodology

This assignment requires a design of the complete Go-Back-N protocol for the datalink layer. The skeleton of this code is given in the Appendix, and provided in the file dlc_layer.c.

- In directory /class/cis677/new_linux/Lab3/Files/, you will find these files:
  - dlc_layer.c: file containing the outline for the lab.
  - Makefile: makefile for the assignment.
  - Four configuration files: Each configuration file specifies a different error rate for the transmission link.
    - 2nodes.config: no errors in transmission.
    - 2nodes_error1.config: the link error probabilities of 0.1.
    - 2nodes_error2.config: the link error probabilities of 0.2
    - 2nodes_error3.config: the link error probabilities of 0.3
      Note: 30% of errors are address errors and the rest are errors in other parts of the frame.
  - lab3_demo: Demo program for this laboratory that you can experiment with.

- Copy the above files to your directory.
- Experiment with lab3_demo.
- Study Go-Back-N-ARQ and these instructions carefully.
- Now you are ready to write your program for the datalink layer. You have to fill in the appropriate code in dlc_layer.c; don't worry about the configuration files.
- To compile your program, type make. This will produce an executable called lab3_exec in your working directory.
- Now execute your version of the code and use the configuration files to make sure it works.

5. Submissions

- You must submit your source code file dlc_layer.c electronically by the command:
  
  submit c3461ax lab3 dlc_layer.c

  where ‘x’ depends on the class section you are in.

- You must also submit hard copies of the following:
  - a. A short summary of the laboratory
  - b. Your source code.
#include "cisePort.h"
#include "sim.h"
#include "component.h"
#include "comptypes.h"
#include "list.h"
#include "eventdefs.h"
#include "main.h"
#include "route_activity.h"
#include "sim_tk.h"
#include "dlc_layer.h"

static int
window_open(DLC_Conn_Info_TYPE *dci)
{
    int result;
    /* Based on a number of a_pdu's in the transmission buffer and values for snd_nxt, snd_una and window_size determine if there is a new a_pdu ready to be sent */
    return result; // result = 1, there is an a_pdu ready
    // result = 0, there is not
}

static
dlc_layer_receive(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
        GENERIC_LAYER_ENTITY_TYPE *generic_layer_entity,
        PDU_TYPE *pdu)
{
    /* Gets the appropriate DLC_Conn_Info_TYPE structure */
    DLC_Conn_Info_TYPE *dci;
    dci = Datalink_Get_Conn_Info(dlc_layer_entity, pdu);

    if (DatalinkFromApplication(generic_layer_entity)) { 
        FromApplicationToDatalink(dlc_layer_entity, pdu, dci);
    } else if (DatalinkFromPhysical(generic_layer_entity)) {
        FromPhysicalToDatalink(dlc_layer_entity, pdu, dci);
    }
    return 0;
}
/**************************************************************/
/* YOU MUST HAVE THIS FUNCTION */
static
FromApplicationToDatalink(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
PDU_TYPE *pdu_from_application,
    DLC_Conn_Info_TYPE *dci)
{
    /* Insert the pdu from the application layer to the
    transmission buffer. */
    /* If possible send info frame. */
}

/**************************************************************/
/* YOU MUST HAVE THIS FUNCTION */
static
FromPhysicalToDatalink(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
PDU_TYPE *pdu_from_physical,
    DLC_Conn_Info_TYPE *dci)
{
    /* Check and discard the pdu when error */
    /* If not discarded, check d_pdu.type and call an
    appropriate function:
    if RR you may use and call DataLinkProcessRR()
    if REJ you may use and call DataLinkProcessREJ()
    if INFO you may use and call DataLinkProcessINFO() */
}

/**************************************************************/
/* YOU DO NOT HAVE TO HAVE THIS FUNCTION */
static
DataLinkProcessRR(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
PDU_TYPE *pdu,
    DLC_Conn_Info_TYPE *dci)
{
    /* If this is a response RR with P/F_bit=0:
    Free up space in the retransmission buffer. Use:
    UpdatePDUBuffer(dlc_layer_entity,pdu,dci);
    Update snd_una
    Send as many info pdu's as allowed by window. You may
    use: window_open(dci) and SendInfo(dlc_layer_entity, dci);
    */

    /* If this is a command RR with P/F_bit=1:
    Free up space in the retransmission buffer.
    Clear rej_already_sent
    Update snd_una
    Send as many info pdu's as allowed by window.
    Create and send to physical layer a response RR with P/F=1
    */

    /* If this is response RR with P/F_bit=1 and rr_pbit=1:
Free up space in the retransmission buffer.
Clear rr_pbit_sent

Update snd_una and snd_nxt
Send as many info pdu's as allowed by window.

/

/* Ignore and discard all other RR's */
/* Free pdu */
return 0;

/***************************************************************************/
/* --- YOU DO NOT HAVE TO HAVE THIS FUNCTION --- */
static DatalinkProcessREJ(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
PDU_TYPE *pdu, DLC_Conn_Info_TYPE *dci)
{
/* Check the address and P/F_bit and if not correct discard frame
Otherwise, free up space in the retransmission buffer
update snd_una and snd_nxt
Send as many pdu's as allowed by window
*/
/* Free pdu */
return 0;

/***************************************************************************/
/* --- YOU DO NOT HAVE TO HAVE THIS FUNCTION --- */
static DatalinkProcessInfo(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
PDU_TYPE *pdu, DLC_Conn_Info_TYPE *dci)
{
PDU_TYPE *pdu_to_application;

/* Check the address and P/F_bit and if not correct discard frame */
/* Check if the pdu has the expected sequence number
When out of sequence, then send REJ, discard pdu and
return 0 */

/* If expected PDU, then increment rcv_nxt,
clear rej_already_sent and RR is sent
You may use SendRR() to send RR to physical */

/* --- Send pdu to application */
/* Free pdu */
return 0;

/***************************************************************************/
/* --- DO NOT CHNAGE NAME OF THIS FUNCTION --- */
/* The function is automatically called when the timer expires */
static
DatalinkTimerExpired(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
                   DLC_Conn_Info_TYPE *dci)
{
    /* Only if there are a_pdu in the buffer, allocate a new d_pdu
       and fill in the needed fields for a command RR with P_Fbit=1
       Set rr_pbit_sent=1
       Send to d_pdu to physical layer */
    return 0;
}

/**************************************************************/

/***** YOU DO NOT HAVE TO HAVE THIS FUNCTION ***/
static
SendInfo(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
          DLC_Conn_Info_TYPE *dci)
{
    /* get a_pdu to send */
    /* Copy it to a new d_pdu and fill the remaining fields */
    /* increment snd_nxt */
    /* Set timer */
    /* Send d_pdu to physical layer */
    return 0;
}

/**************************************************************/

/***** YOU DO NOT HAVE TO HAVE THIS FUNCTION ***/
static
SendRR(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity, PDU_TYPE *pdu,
        DLC_Conn_Info_TYPE *dci)
{
    /* Allocate a new d_pdu and fill in the needed fields */
    /* Send to d_pdu to physical layer */
    return 0;
}

/**************************************************************/

/***** YOU DO NOT HAVE TO HAVE THIS FUNCTION ***/
static
SendREJ(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity, PDU_TYPE *pdu,
         DLC_Conn_Info_TYPE *dci)
{
    /* Don't send REJ nor RR if rej_already_sent = 1 */
    /* If REJ is to be send, allocate a new d_pdu and fill in the
       needed fields */
    /* Send to d_pdu to physical layer */
/* rej_already_sent set to 1 */

return 0;
}