Faculty of Computing, Engineering and Mathematical Sciences

Internal Moderation of Coursework

General Instructions

The module leader should supply to the moderator a printed copy of the proposed assignment which should include indications of due date, weighting, assessment criteria and effort required. The moderator should use this form to comment on and progress the assignment, recording brief comments on the form and more extensive ones on the assignment specification. When the assignment has been agreed with the module leader, the moderator should sign the form and submit it along with the assignment to the CEMS Programmes Office.

Module Leader to complete this Section

module name **CSA**

module number **ufeEHF-30-1**

assignment number **2**

issue date **6/2/08**

% weighting in module **35%**

estimated time to complete **18 hrs**

module leader **Rob Williams**

internal moderator **Nigel Gunton**

work set by **Rob Williams**

Moderation

The moderator should check the assignment is satisfactory with respect to:

- rubric (including due date, weighting, and estimated effort)
- the task specification
- mark allocation and assessment criteria
- level of work
- effort needed

moderator's comments

setter's response

R Williams

Internal moderation completed

date

signed
(internal moderator)

MODULAR PROGAMME
ASSESSED COURSEWORK SPECIFICATION

Module Details:

Module Code: ufeEHF-30-1	Module Title: Computer Systems Architecture	
Module Leader: Rob Williams		
Module Tutors: Ian Anderson John Counsell Laurence O'Brien		
Assignment CW2	Element Number: Weighting 35%	Total Assignment Time: 12 hrs

Dates:

Date assignment issued to students: Feb 4th	Date for return of marked work: May 8th
Submission Place: postbox in N foyer, below the North stairs	Date of Submission: Thurs 10th April
	Time of Submission: 10.00am

Deliverables:

As listed on the Assignment spec sheet
Work in pairs, submitting a single report

BSc CSI/CRTS/CSE, ufeEHF-30-1, CSA Assignment 2 (Feb 08)

Hand-in date: Thurs 10th April

This work is an extension of the first assignment and the recent worksheet on RS232 Serial Communications. It serves as an introduction to networking.

You are required to work in pairs (**two** people), only a single piece of work is to be submitted for assessment and each contributor will receive the same mark. Working in threes or more is not permitted unless agreed by the module leader (Rob Williams).

The aim is to produce a semi-robust text messaging system to pass packets of data between small clusters of PCs. The PCs will be connected by a special looped cable through one of their COM ports. The packets must conform to the following construction:

Packet Structure

```
  ___________________________________________________________
 |   |   |   |   |              10 Byte              |     |   |
 | { | D | S | T | <-----      Payload    ------>    | CS  | } |
 |___|___|___|___|___________________________________|_____|___|
                      16 bytes
 Packet head                                              comes
 comes first                                              last
```

{ - start marker

D - user ID at destination station. A-Z

S - user ID at source station. A-Z

T - packet type:

- L - login by new user
- X - logout by user
- R - response to login packet
- D - data payload
- Y - acknowledge, ACK, good packet received
- N - nonacknowledge, NAK, damaged packet received

Payload - The 10 character messages originate from the keyboard. Local messages are displayed on the screen and acknowledged to the source. Nonlocal messages are simply passed onward without display.

CS - the single byte checksum should be the inverted modulo-128 sum of all the other bytes in the packet. This is a 7 bit number but to avoid confusion with other ASCII chars, set the ms bit during transmission. If a packet is received with a good checksum, an ACK packet is returned, otherwise a NAK packet. ACK & NAK packets themselves are never acknowledged. The implementation of this facility may be left until the end.

} - end marker

Functionality
Data is transmitted at 9600 bps, 8 bits, no parity, 1 stop bit. Hardware flow control (CTS/RTS) must be enabled. This can be done using Hyperterminal/minicom during the hardware checking procedure, but should be repeated by initialization code within your own program.

Users are to be uniquely identified by a SINGLE letter (A-Z) which must be entered at LOGIN by the current user. When a user logs out the local ID is set to zero.

Stations with no user logged in must still pass packets and not interrupt the loop.

On boot-up and before user log in, the station will have its user ID set to zero. When a new user logs in, an L packet is transmitted with the source and destination fields set to the new ID. No ACK packets should be received from this and the L packet should return home for destruction. Any intermediate stations with a logged in user, should send an R packet to the new user, informing of their presence. This is to ensure that the chosen ID is unique for the ring. If an ID duplication has occurred, the other station will see its own ID, remove the packet and transmit an ACK packet back. Should this happen, an error message should be displayed to the new user, and an alternative ID be requested.

The station should be waiting for incoming packets and at the same time watching the keyboard for input.

A Send Message keyboard sequence starts with a ’D’ (destination). A valid ID letter is then requested after which the next character string, upto 10 characters, or a RET, is the message. The new message is displayed and then sent if an ’S’ is entered.

Packets to unknown users should be refused before transmission, either using a probe transmission or, better, by implementing a local Active User List.

L packets can help support local User Lists of active users. When a station passes an L packet through, it will also respond with an ’R’ packet to allow the new user to find out who else is logged in. The array. This directory can then be maintained by spying on passing traffic, and acting on LOGOUT ’X’ packets.

To logout from the system, the letter ’L’ should be used. This is not a system closedown, packets should continue passing through the station.

Packets with the local user ID as source should be deleted. Packets with the local user ID as destination are displayed, ACKed and deleted. All incoming damaged packets must be deleted and not retransmitted.

When a packet is transitted it should be stored (pended) in case it needs to be retransmitted in the event of a NAK or more commonly a time-out error (failure to receive an ACK within 5 secs). After 4 retransmission attempts, an error message should be displayed and the destination ID deleted from the local directory. A proxy LOGOUT packet might then be transmitted to tell the other stations of the nonexistence of that ID.

Advice & Hints
1. Read this spec, read it again. And again.
2. Attend briefing lectures, and read the spec. Again.
3. Your software should be structured as three tasks or threads based on the FSD examples provided. These will handle: Packet Reception, Packet Transmission, Keyboard & Screen.
4. Understand the desired functionality. Then sketch a s/w design. Then start coding incrementally so as to be able to test each part.
5. A helpful Debug mode should be provided (ˆD toggle on/off) to display the full incoming packet structure. Normally only the payload message and source ID should be displayed.
6. To avoid the situation of two users simultaneously logging-in with the same ID, the L packet payload should contain a unique key, such as local time (`GetTickCount()` on Windows or `times()` on Linux).

Getting started
Start with only two PCs interconnected and check an end-to-end serial link with Hyperterminal. Kill one Hyperterminal, and implement the given kbd starter code. Login and watch Hyperterminal at the receiver end. You may respond by hand typing a reply packet from Hyperterminal. If all seems to be working OK, kill off Hyperterminal and run up `mirror.exe` in its place. This displays received packets and reflects them back, too. Now work on developing your Tx & Rx tasks, using `mirror.exe` as a debugging aid. Use the VisualStudio debugger to single step through packet transmit and receive sequences. This takes time but builds your confidence in the code. For home development, you can use a single PC if it has two COM ports. Good luck!

Deliverables

Source Code in C **40%**

a) Comments focused on the problem not the instructions

Comments not over long, so unmaintainable

Program banner: author name & date, revision date, functional description, user advice

Function banner comments, functional description, parameter list, warnings

Clear code structure expressing three tasks with operational sequences

(possibly using finite state implementation through SWITCH/CASE or table)

Device opens are error checked

Functions use parameters effectively

Debug mode to optionally display all packets

Non-blocking code structure

Computes packet 7 bit checksum for tx, and validates checksum on rx

Requests retransmission using a NAK if checksum error detected

b) Ability to check that new login ids are currently unique

Builds online user list

Maintains current user list

Refuses messages to invalid user ids but can send test message to self

c) Pends outgoing packets for re-tx, clears pending packets on ACK, or after 5 attempts

Demonstration 1 station with the mirror.exe test code. **(5% max)**

OR

Demonstration with only 2 stations. **(15% max)**

OR

Demonstration with more than 2 stations. **30%**

boots up from a desktop icon and starts running

accepts user login ID letter from keyboard

accepts keyboard message (<=10 char or CR) and destination ID

transmits message packet to mirror.exe

displays debug message indicating incoming packet and full contents

displays message payload when correct destination (self-addressed)

removes packet (because of source ID)

allows user to logout without rebooting system

passes packets onward before user logs in

Checks new ID is unique on the LAN

Refuses poorly addressed packet with error message

Does not block during keyboard message input

Basic messaging functionality available

Basic messaging functionality available

Stores outward packet until acknowledge received and retransmits a limited number of times.

Will not send a packet to an unresponsive station.

Recovers from a pathological test sequence

Full finite state diagrams for each part of your system **15%**

Accurately describes the functionality for transmission, reception, keyboard entry and packet handling. The FSDs must be easily related to the code (or table).

The URL for a small website (2 pages) **15%**

This advertises your product, describing its functionality to potential users.

Provide your executable code as a zip file for downloading and installation

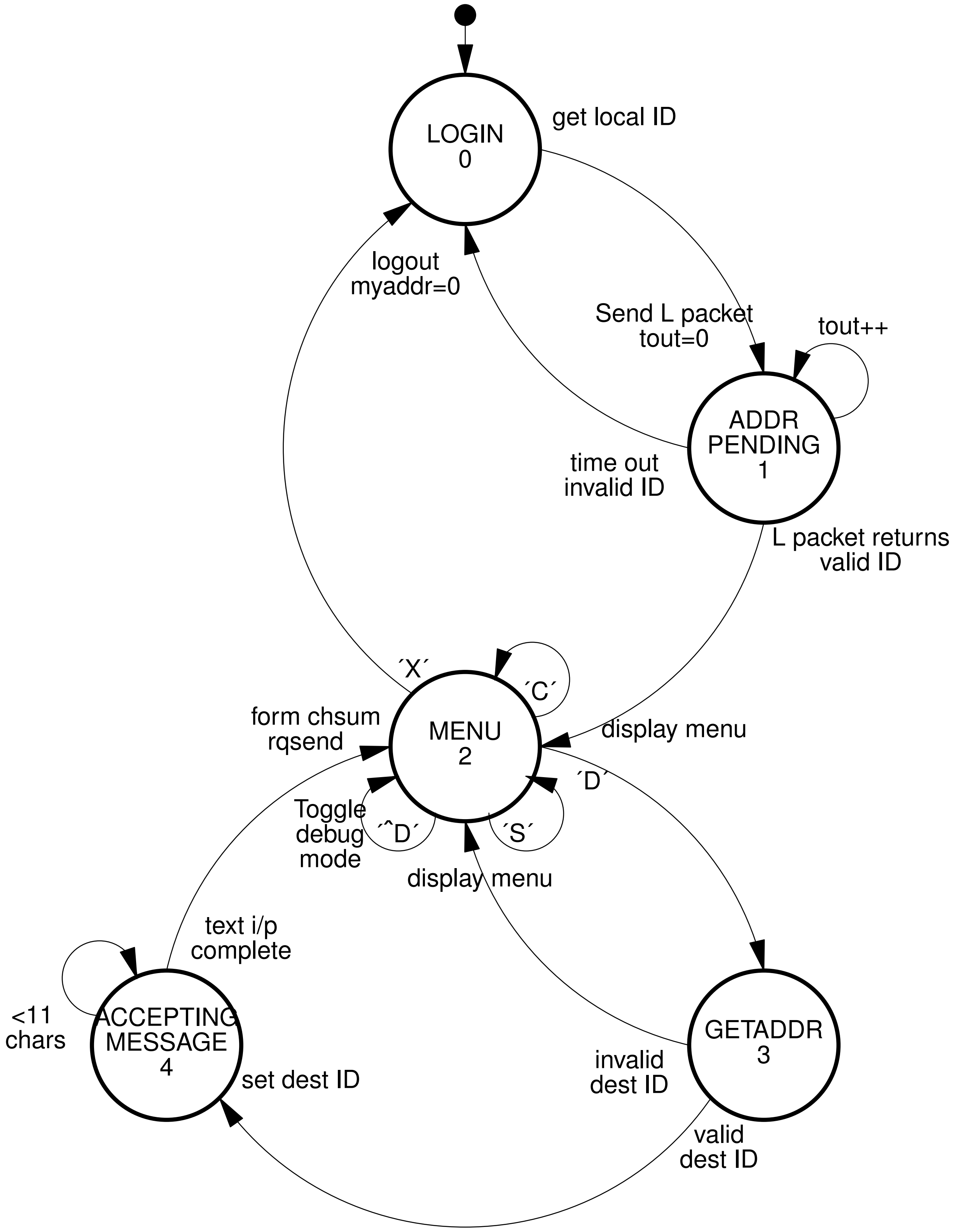
There should be installation instructions and user guidance

BONUS marks:

For interworking reliably with someone else's program. **5%**

For interworking reliably with foreign language code. **5%**

For interworking reliably with foreign operating system. **5%**



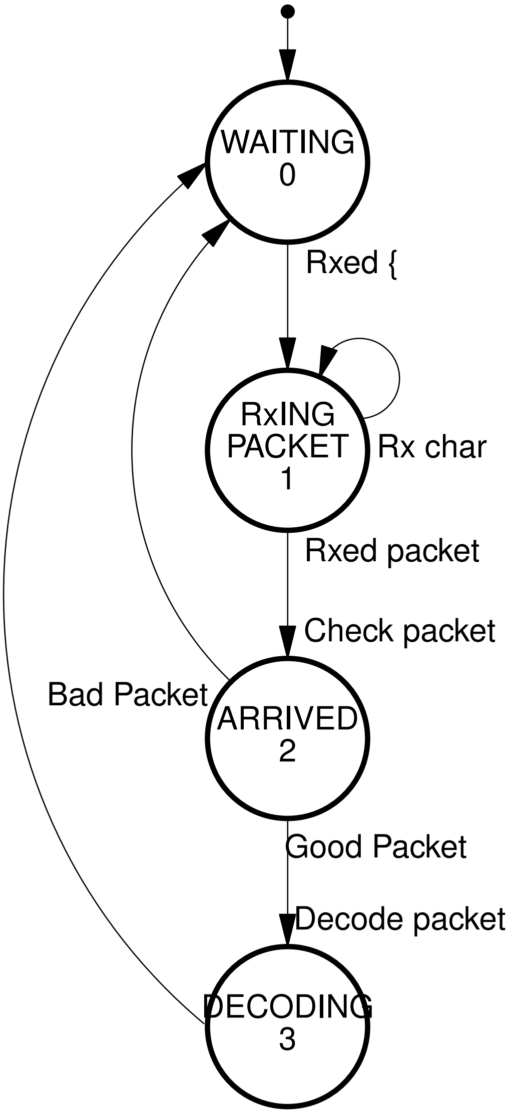
FSD for Keyboard handler Task

```
/* Rob Williams Feb 5th 2008
   Starter code for RingLAN keyboard task to run on Windows. Contains some KBD & TX
   but no Rx code. Also the tx packet pending facility is not yet fully implemented here.
*/
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <windows.h>
#include <winbase.h>

#define LOGIN 0          // states for KBD FSM
#define ADDRPENDING 1
#define MENU 2
#define GETADDR 3
#define INPUTMESS 4

#define WAITING 0        // states for RX FSM
#define RECEIVING 1
#define ARRIVED 2
#define DECODING 3

char myaddr = 0;
int rqsend;
int rxflag;
int kindex, rxindex, txindex;
DWORD dwError;

char  rxpacket[16];
char  txpacket[16];
char kbdpacket[16];

//  { dest src type <------data--------->  cs }
//  0  1    2   3   4                   13 14 15

struct z {
     int loggedin;       //0: not logged in, 1: logged in, -1: home
     int pending;        //packet pending?
     char packet[16];    //last packet txed
} pendtable[26];

int pendindex = 0;

HANDLE hCom;
BOOL fSuccess;

/* Initializes serial port
*  entry parameter: pointer to device name string
*  exit: sets up rx/tx parameters, no blocking
*        sets global hCom
*/
void initcomm(char* device) {  //--------------------------------------

 COMMTIMEOUTS noblock;
 DCB dcb;

 hCom=CreateFile(device,
      GENERIC_READ | GENERIC_WRITE,
      0,
      NULL,
      OPEN_EXISTING,
      0,
      NULL
      );
     if (hCom == INVALID_HANDLE_VALUE) {
      dwError = GetLastError();
      printf("INVALID_HANDLE_VALUE()");
     }

 fSuccess = GetCommTimeouts(hCom, &noblock);
     noblock.ReadTotalTimeoutConstant = 1;
     noblock.ReadTotalTimeoutMultiplier = MAXDWORD;
     noblock.ReadIntervalTimeout = MAXDWORD;
 fSuccess = SetCommTimeouts(hCom, &noblock);

 fSuccess = GetCommState(hCom, &dcb);
 if(!fSuccess){
      printf("GetCommState Error!");
 }
     dcb.BaudRate = 9600;
     dcb.ByteSize = 8;
     dcb.fParity = FALSE;
     dcb.Parity = NOPARITY;
     dcb.StopBits = TWOSTOPBITS;
     dcb.fRtsControl = RTS_CONTROL_HANDSHAKE;
     dcb.fOutxCtsFlow = TRUE;
 fSuccess = SetCommState(hCom, &dcb);
 if(!fSuccess){
      printf("SetCommState Error!");
 }

 printf("Comm port set\n");
}

/* Nonblocking read from serial port identified by global hCom
*  returns: char or 0
*/
char readcomm()               //----------------------------------------
{
      char item;
      int ni;
 fSuccess = ReadFile( hCom,
          &item,
          1,
          &ni,
          NULL
   );
  if (ni >0 ) return item;
  else return 0;
}

/* Nonblocking read from keyboard
*  returns: char or 0
*/
char readkbd()                //----------------------------------------
{
 if (kbhit() ) return getch();
 else return 0;
}

/* Sets up a fresh packet ready for use
*  entry: pointer to 16 byte area for packet
*/
void clearpacket(char* ppacket) {//----------------------------------------
int i=0;
     ppacket[i++] = '{';
     ppacket[i++] = 0;
     ppacket[i++] = myaddr;
     ppacket[i++] = 0;
     while ( i < 15) ppacket[i++] = ’ ’;
     ppacket[i] = ’}’;
}

/* Calculates and sets a checksum
*  entry: pointer to a packet
*/
void setchsum(char * ppacket) {  //--------------------------------------
int chsum = 0, i;
     ppacket[14] = 0;
     for (i=0; i<16; i++) chsum += ppacket[i];
          ppacket[14] = ~ (chsum%128);
          ppacket[14] |= 0x80;
}

/* dispatches packet to port for transmission
*  entry pointer to 16 byte packet
*
*/
int sendpacket(char* ppacket) {  //--------------------------------------
      int ni;
 fSuccess = WriteFile( hCom,
          ppacket,
          16,
          &ni,
          NULL
   );
  if (ni == 16 ) return 0;
  else return 1;

}
/* A starter program using 3 sequential, non-blocking, cooperative tasks
*  this is only one solution, there are many others, such as pthreads.
*/
void main() {                    //--------------------------------------

int confirm = 0;
int key, i, keycnt=0;

kindex = LOGIN;
myaddr = 0;
initcomm("COM1");
puts("Welcome to the text ring, plz enter your id\n");

while (1)
{
/********************************* KbdTask ************************************************/
switch (kindex) {

case LOGIN:
          if (kbhit()) {
               key = toupper(getch());
               if ( key >= ’A’ && key <= ’Z’) {
                    myaddr = key;
                    clearpacket(kbdpacket);
                    kbdpacket[1] = myaddr;
                    kbdpacket[2] = myaddr;
                    kbdpacket[3] = ’L’;
                    setchsum(kbdpacket);
                    for(i=0; i<26; i++) {               // clear pending table
```

```
                       pendtable[i].loggedin = 0;
                       pendtable[i].pending = 0;
                    }
                    strncpy(pendtable[myaddr-’A’].packet, kbdpacket, 16);
                    pendtable[myaddr-’A’].pending = 5;   // 5 attempts to login!
                    kindex = ADDRPENDING;
               }
          }
break;
case ADDRPENDING:
     if (pendtable[myaddr-’A’].loggedin == -1) { //id OK for me, temp test code, RX will do thi
               printf("Your home id is now set to: %c\n", myaddr);
               kindex = MENU;
          } else {
               if (pendtable[myaddr-’A’].pending < 1) {
                    puts("Either cable break or duplicate login id, try again\n");
                    kindex = LOGIN;
                    pendtable[myaddr-’A’].pending = 0;
                    myaddr = 0;
               }
          }
break;
case MENU:
     puts("Options: Destination, Send, Cancel, Logout\n");
     if (kbhit()) {
          key = toupper(getch());
          switch (key) {
          case ’D’:
               puts("The destination address is: ");
               kindex = GETADDR;
          break;
          case ’S’:
               if (kbdpacket[1])  {
                    pendtable[kbdpacket[1]-’A’].pending = 5;
                    clearpacket(kbdpacket);
               }
               break;
          case ’C’:
               clearpacket(txpacket);
          break;
          case ’L’:
          case ’Y’:
               if (!confirm) {
                    puts("\nAre you sure you mean to logout? Y/N\n");
                    confirm++;
               } else {
                    puts("\nLogging you out now\n");
                    confirm = 0;
                    clearpacket(kbdpacket);
                    kbdpacket[1] = myaddr;
                    kbdpacket[2] = myaddr;
                    kbdpacket[3] = ’X’;
                    setchsum(kbdpacket);
                    if(pendtable[myaddr-’A’].pending == 0) {
                         strncpy(pendtable[myaddr-’A’].packet, kbdpacket, 16);
                         pendtable[myaddr-’A’].pending = 1;
                    }
                    for(i=0; i<26; i++)
                         pendtable[i].loggedin = 0;
                    myaddr = 0;
                    kindex = LOGIN;
               }
          break;
          default:break;
          }
     }
break;
case GETADDR:
     if(kbhit()) {
          key = toupper(getch());
               if(pendtable[key-'A'].loggedin==0) {
                    puts("Destination not logged in at present\n");
                    kindex = MENU;
               }else {
                    clearpacket(kbdpacket); //set up packet
                    kbdpacket[1] = key;     //dest addr set
                    keycnt=4;
                    kindex = INPUTMESS;
               }
     }
break;
case INPUTMESS:
     if(kbhit()) key = getch();
    if( keycnt<14 && key != '\n')
          kbdpacket[keycnt++] = key;
     else {
          setchsum(kbdpacket);
          if(pendtable[kbdpacket[1]-'A'].pending == 0) {
               strncpy(pendtable[kbdpacket[1]-'A'].packet, kbdpacket,16);
               kindex = MENU;
          } else {

          }
     }
break;
}

/********************************** TxTask *************************************************/
/* Scans down through the pending table looking for packets to transmit                   */

{
     if (++pendindex > ('Z'-'A')) pendindex = 0;
     if (pendtable[pendindex].pending > 0) {
       sendpacket(pendtable[pendindex].packet);   // transmit next available packet
       pendtable[pendindex].pending--;
     }

}
/********************************** RxTask *************************************************/

 switch (rxindex) {
     case WAITING:
     break;
     case RECEIVING:
     break;
     case ARRIVED:
     break;
     case DECODING:
     break;
     default:
     break;
 }
 }//forever, 3 task loop
} //main
```

A	0/1	0-5	packet pending Tx
B			
Y			
Z			

The `pendtable` is a significant data structure at the centre of the operation. It is structured to record details of who is logged into the RingLAN, and also to hold copies of packets due for transmission (first or subsequent times). So it acts as a directory and a transmission queue. The first field is initialized to 0, and set to 1 when that letter/id is used following a login. It also helps to use -1 to mark the home station id. The second field is used to indicate that a packet is waiting for transmission to that station. As 5 retries are required before giving up, the field can be set to 5 and decremented on each transmission attempt. With this setup, multiple simultaneous retries can be handled, but with only one packet for each station.



FSD for Rx Packet Task

Using a decision table when considering all the flavours of incoming packets which have to be catered for is possibly better than resorting immediately to a FSD. Consider the following table which describes all the combinations of the different field values in a packet..

Src	Des	Pkt Type	Action
me	me	L	my login id is OK, del packet
		R	illegal
		D	test mesg, del packet, return ACK
		A	cancel pending entry, del packet
		N	retransmit pending table entry
		X	logout now, del myaddr & pending table
	you	L	illegal
		R	error, where has he gone now?
		D	rx failed, del packet, re-tx from pending table
		A	rx failed, del packet,
		N	rx failed, del packet,
		X	illegal
you	me	L	illegal
		R	response to my login, del packet, update pending tbl
		D	real mesg!, return ACK, del packet
		A	del packet, cancel pending entry
		N	del packet, re-tx from pending table
		X	illegal
	you	L	re-tx packet, update pending, return R
		R	illegal
		D	re-tx packet
		A	re-tx packet
		N	re-tx packet
		X	logout info, re-tx packet, ammend pending table

The decision table then needs to used when coding the rx packet handler task.

```
if (rxpacket[1]==myaddr) {
       if (rxpacket[2]==myaddr) {
          switch (rxpacket[3]) {   // to me from me
            case ’L’:
              ...........;         // my login id is OK, del packet
            break;
            case ’R’:
              ...........;         // illegal
            break;
            case ’D’:
              ...........;         // test mesg, del packet, return ACK
            break;
            case ’A’:
              ...........;         // cancel pending entry, delete packet
            break;
            case ’N’:
              ...........;         // retransmit pending table entry
            break;
            case ’X’:
              ...........;         // logout now, del myaddr & pending table
            break;
            default:
            break;
          }
       } else {
                                   //to me from you

       .....repeat SWITCH/CASE...........


       }
} else {
       if (rxpacket[2]==myaddr) {
                                   //to you from me

       .....repeat SWITCH/CASE...........


       } else {
                                   // to you from you

       .....repeat SWITCH/CASE...........


       }
}
```

This code gets a bit repetitive, but is is easy to understand, with the aid of coloured pens to highlight the blocks of code! A better scheme might use a jump table to select the appropriate code paragraphs.