Xarxes de Comunicació

Pràctica 2 - Protocol d'accés al medi / Xarxa d'àrea local

Francisco del Águila López

Setembre 2022

Escola Politècnica Superior d'Enginyeria de Manresa
Universitat Politècnica de Catalunya

1 Objectiu

L'objectiu d'aquesta pràctica és definir un mòdul en C que implementi un protocol no fiable en el nivell d'enllaç quan el medi és un canal comú compartit.

2 Introducció teòrica

2.1 Direccionalitat de les comunicacions

Els tipus de comunicació entre dos entitats, des del punt de vista del sentit cap on van les dades, es poden classificar de la següent manera:

Simplex: La comunicació es dóna en un únic sentit. Una entitat és exclusivament transmissora i l'altra és exclusivament receptora. Un exemple podria ser la radio / televisió comercial.

Half-duplex: En aquest cas la comunicació és en els dos sentits. Les dues entitats són transmissores i receptores però la restricció és que no es poden donar simultàniament. El principal motiu que això sigui així és perquè es fa servir un únic canal de comunicació per als dos sentits. Un exemple és la comunicació amb radiotransmissors.

Full-duplex: La comunicació es dóna en els dos sentits i pot ser simultània. La manera més simple d'aconseguir-ho és disposar de dos canals independents, un per

cada sentit. També es pot aconseguir amb un únic canal sempre que hi hagi un mecanisme per separar adequadament els dos sentits, com per exemple el telèfon.

En el cas de la comunicació Half-duplex és important conèixer en quin moment l'altra entitat no està transmeten per determinar que el canal està lliure. La detecció de canal de comunicació lliure és de vital importància a les comunicacions on moltes entitats comparteixen un únic canal de comunicació (xarxes locals).

2.2 Fiabilitat de les comunicacions

Des del punt de vista de la fiabilitat d'un enllaç, les comunicacions poden ser:

no fiables: Les dades es transmeten sense la garantia que arribin al destinatari. Aquestes dades es poden perdre o bé ser rebudes amb errors. No hi ha cap mecanisme que detecti o controli això.

fiables: En aquest cas existeix un mecanisme que assegura que les dades arriben al destinatari de manera correcta. Per aconseguir-ho s'implementen mecanismes de detecció i control d'errors, incloent la pèrdua de dades. Un mecanisme per aconseguir-ho és basa en la retransmissió de les dades perdudes o errònies.

La implementació d'enllaços fiables obliga a que al missatge transmès se li afegeixi una informació extra que permeti assegurar aquesta fiabilitat. Aquest afegit del missatge pot trobar-se al començament o al final. D'aquesta manera, es crea un nou missatge amb uns camps especials de control que complementen el camp de dades o missatge original.

3 Punt de partida

3.1 Mòdul blck_ether

La natura de la transmissió per àudio o infraroig en Morse fa que el canal de comunicació sigui únic tant en el cas d'una comunicació punt a punt com en el cas de comunicacions en xarxa local. Aquest fet provoca que les comunicacions a nivell d'enllaç tinguin més sentit a nivell de blocs de bytes que a nivell de byte. Transmetre un bloc de bytes de forma compacta facilita la construcció de la trama (unitat de dades d'enllaç), la seva delimitació i el seu posterior tractament. Per aquest motiu la capa física original que oferia servei a nivell de byte es transformarà per oferir servei a nivell de bloc de bytes. Aquesta modificació inicial de la capa física servirà també per ampliar el conjunt de funcions oferides.

Així, el fitxer de capçalera del nou mòdul *blck_ether* és:

```
#ifndef BLCK_ETHER_H
#define BLCK_ETHER_H
#include <inttypes.h>
#include <stdbool.h>
```

```
/* MO-DEM Morse
 * This module use modulator module for MOdulator
 * and implements a DEModulator on PORTD pin2 by
 * interrupt INT0  */

/* Sign durations in ticks */
#define DOT TIMER_MS(80)
#define DASH (DOT*3)
#define GAP DOT
#define CHARGAP DASH
#define LONGGAP (DOT*7)

/* Maximum length of a message */
#define MAX_ME_ETH 126

//Only A..Z, 0..9 and SPACE are valid
typedef uint8_t morse_c_t;

//Callback functions
typedef void (*ether_cll_t)(void);

//Definition of a ether message
typedef morse_c_t missatge_eth_t[MAX_ME_ETH];

//Configuration modes of Ether layer
typedef enum {Normal, InvertedRX} eth_mode_t;
/* InvertedRX: Reception is inverted polarity  */

void ether_init(eth_mode_t mod);

bool ether_busy(void);
// true if ether is transmitting or receiving

bool ether_can_put(void);
// true if ether_block_put() can be called

void ether_block_put(const missatge_eth_t m);
// Function to transmit a message via ether channel

void ether_block_get(missatge_eth_t m);
// Function to receive a message
// after any on_message_received event

void on_message_received(ether_cll_t m);
void on_finish_transmission(ether_cll_t f);

#endif
```

typedef uint8_t morse_c_t És una definició de tipus que contempla els caràcters vàlids que es poden fer servir en un missatge morse. El motiu de la nova definició de tipus és perquè els valors vàlids corresponen als caràcters numèrics 0..9, els caràcters de les lletres majúscules A..Z i el caràcter espai.

void ether_init(eth_mode_t mod) Serveix per inicialitzar el canal físic. Permet rebre el senyal morse o bé el senyal morse invertit. S'ha de tenir en compte que el receptor d'infraroig desmodula el senyal donant el senyal invertit.

bool ether_busy(void) Indica si el canal de comunicacions està ocupat.

bool ether_can_put(void) Es farà servir per comprovar si és possible la transmissió d'un missatge.

void ether_block_put(missatge_eth_t m) Aquesta funció es fa servir per transmetre un missatge. S'ha de considerar que l'indicador de fins a on està ple el missatge ve determinat pel caràcter *nul* ('\0') que farà de sentinella.

void ether_block_get(missatge_eth_t m) Aquesta funció ofereix la possibilitat de llegir el contingut del missatge rebut i el lliura a *m*. El sentinella de final de dades serà igualment el caràcter *nul*.

void on_message_received(ether_callback_t m) Aquesta és una funció que permet instal·lar una funció de callback que serà cridada quan sigui rebut un missatge. Amb aquest callback evitem la necessitat d'haver d'estar contínuament enquestant la rebuda d'un missatge.

void on_finish_transmission(ether_callback_t f) Permet instal·lar un callback que serà cridat quan es produeix la finalització de la transmissió d'un missatge.

Les funcions ether_block_put() i ether_block_get() no retornen res per simplificar, però es podria fer una implementació que retornessin la quantitat real de bytes que han pogut enviar o rebre. Això serviria per poder fer un control dels possibles errors que pugui haver.

3.2 Mòdul serial

Per facilitar la comunicació serie, també s'ofereix aquest mòdul

```
#ifndef SERIAL_H
#define SERIAL_H

#include <inttypes.h>
#include <stdbool.h>

void serial_open(void);
void serial_close(void);
uint8_t serial_get(void);
void serial_put(uint8_t c);
bool serial_can_read(void);

#endif
```

Recordeu que aquest mòdul fa de *driver* de la interfície sèrie. Es disposa d'una *cua* de 127 bytes.

void serial_open(void) Obre i per tant inicialitza les comunicacions sèrie.

void serial_close (void) Tanca i per tant allibera les comunicacions sèrie.

bool serial_can_read(void) Indica si hi ha bytes disponibles per ser llegits en el *buffer* de recepció.

unit8_t serial_get(void) Buida un caràcter del *buffer* de recepció. S'ha de cridar quan hi ha dades disponibles.

void serial_put(uint8_t c) Envia per port sèrie el caràcter que se li passa com a paràmetre.

També s'ofereix el mòdul blck_serial

```
#ifndef BLCKSERIAL_H
#define BLCKSERIAL_H

#include <stdint.h>

uint8_t readblck(char s[]);
/* Read a block until a non printable character is found.
 * Returns the length.
 */

void print(char s[]);

#endif
```

3.3 Mòdul timer

Aquest mòdul és el mateix que es planteja a la pràctica de "Control semafòric de cruïlla amb comunicació morse: mestre" de l'assignatura de Programació de Baix Nivell. El fitxer de capçalera és el següent

```
#ifndef TIMER_H
#define TIMER_H

/*
 * This module implements a time dispatcher with a resolution
 * of 5 ms. It is based on callbacks. That is, functions which
 * are called after a specific (temporal) event ocurred.
 */
#define TIMER_MS(ms) (ms/5)
#define TIMER_ERR    -1

typedef void (*timer_callback_t)(void);
typedef int8_t timer_handler_t;
typedef int8_t timer_chrono_t;

void timer_init(void);
void timer_cancel(timer_handler_t h);
void timer_cancel_all(void);
timer_handler_t timer_ntimes(uint8_t n, uint16_t ticks, timer_callback_t f);
timer_handler_t timer_every(uint16_t ticks, timer_callback_t f);
timer_handler_t timer_after(uint16_t ticks, timer_callback_t f);

timer_chrono_t chrono(void);
void chrono_start(timer_chrono_t c);
uint16_t chrono_get(timer_chrono_t c);
void chrono_stop(timer_chrono_t c);
void chrono_cancel(timer_chrono_t c);

#endif
```

S'ofereix un servei de temporització amb aquest mòdul. Essencialment consisteix en executar una funció f() planificant la seva execució per d'aquí a k ms. El nombre màxim de ticks és un uint16_t que amb una resolució de 5ms dóna un temps màxims de 327 segons.

void timer_init(void) Inicialitza el mòdul. Cal cridar-la com a mínim una vegada abans d'usar el mòdul. Només pot cridar-se amb les interrupcions inhabilitades.

void timer_cancel(timer_handler_t h) Cancel·la l'acció planificada identificada per h. Si h no és un handler vàlid, no fa res.

void timer_cancel_all(void) Cancel·la totes les accions planificades del servei.

timer_handler_t timer_after(uint16_t ticks, timer_callback_t f) Planifica la funció f() per ser executada al cap de ticks ticks. Retorna un handler que identifica

aquesta acció planificada o bé val TIMER_ERR en cas que l'acció no es pugui planificar per alguna raó.

timer_handler_t timer_every(uint16_t ticks, timer_callback_t f) Planifica la funció f() per a ser executada cada ticks ticks de manera indefinida.

timer_handler_t timer_ntimes(uint8_t n, uint16_t ticks, timer_callback_t f) Planifica la funció f() per a ser executada cada ticks ticks n vegades. En cas que n sigui zero s'interpreta que la funció ha de ser cridada indefinidament.

El mòdul també ofereix les implementacions de cronòmetres.

3.4 Mòdul error_morse

Aquest mòdul és el que s'ha desenvolupat a la pràctica anterior.

3.5 Generació de número aleatori

En aquest protocol d'accés al medi hi ha la necessitat d'esperar un temps aleatori. Per generar aquest temps es farà servir la funció rand() de la llibreria <stdlib.h>. Mireu la documentació a [avr-libc].

4 Protocol d'accés al medi

El protocol que s'ha d'implementar és un CSMA (accés múltiple amb detecció de portadora). Per tant, si en el moment que una estació vol transmetre es detecta que el canal està ocupat s'esperarà un temps aleatori i ho tornarà a intentar. Si detecta canal lliure realitzarà la transmissió.

Els nodes de la xarxa local seran tant transmissors com receptors.

No es farà cap tipus de control dels errors. Per tant aquesta capa oferirà un servei no fiable. L'única gestió que es farà amb els erros és que si es detecta una trama errònia es descartarà aquesta trama directament.

Existeix la necessitat d'identificar tots els nodes existents a la xarxa local. Per resoldre aquest problema s'assignarà a cada node una adreça corresponent a un caràcter morse. D'aquesta manera amb un únic byte identificarem als nodes. Es recorda que els caràcters morse és l'abecedari en majúscules i els números del 0 al 9.

4.1 Recepció de missatges

El comportament que ha de tenir el protocol en el moment de rebre un missatge és el següent:

- Analitza si el missatge rebut (trama) és un missatge vàlid. Això ho farà comprovant els bits de redundància. Si no és vàlid el descarta i no fa res més.
- Analitza si el destinatari del missatge és ell. Si no és així descarta el missatge i no fa res més.
- Finalment extreu les dades i l'adreça d'origen i lliura aquestes dades a la capa superior per mitjà de l'execució del callback que se li ha passat via on_message_received().

4.2 Transmissió de missatges

El comportament del protocol per transmetre un missatge és el següent:

- Un missatge per transmetre arriba quan la capa superior fa una crida a lan_block_put(). Quan això passa, primer construeix la trama que ha d'enviar. Testeja si la pot transmetre. En cas positiu el transmet i acaba.
- En cas negatiu calcula un temps aleatori entre 1 i 10 segons, i activa un esdeveniment temporal (amb el mòdul timer) per tornar a fer l'intent de transmissió passat aquest temps aleatori.
- Repeteix aquest procés indefinidament fins que aconsegueix la transmissió. Queda delegat a l'usuari resoldre el problema en el cas que la comunicació no es pugui dur a terme mai.

4.3 Unitat de Dades de Protocol

La unitat de dades de protocol (PDU) en aquest cas rep el nom de trama. Un dels aspectes més rellevants en la especificació d'un protocol és la definició de com han de ser aquestes trames. En aquesta pràctica només hi ha un tipus de trama, ja que no hi ha necessitat de trames de control.

La trama està formada pels següents camps:

Adreça d'origen És el camp corresponent a identificar el node propi. El valor d'aquest camp és el d'un caràcter morse. S'han de gestionar adequadament aquestes adreces per evitar la duplicitat d'adreces entre diferents dispositius a la mateixa xarxa local.

Adreça de destí És el camp corresponent a identificar el node a qui se li envia la informació i consisteix en un caràcter morse.

Camp de dades És un camp de mida variable múltiple de Byte, que contindrà les dades que s'han de transportar. En general, contindrà la unitat de dades de protocol de la capa superior, però en el cas de la pràctica, la capa superior directament serà la capa d'aplicació, per tant contindrà els missatges que es volen transmetre.

Camp de FCS És un camp que ocupa 2 caràcters. Conté el Checksum / CRC calculat segons la pràctica anterior. Per conveni es considera que el caràcter (byte) de

més pes s'envia primer i el segon caràcter és el de menys pes. FCS: Seqüència de Comprovació de Trama. En el cas particular del protocol que estem implementant, es fa servir l'algoritme de CRC.

Per poder modelar aquesta unitat de dades de protocol amb el llenguatge C, es proposa la següent definició de tipus:

```
#define MAX_PAYLOAD_LAN (MAX_ME_ETH - 2) //2 bytes of addresses

typedef struct {
   morse_c_t origen;
   morse_c_t desti;
   morse_c_t payload[MAX_PAYLOAD_LAN];
} lanpdu_t;
```

S'ha de tenir en compte que el camp payload[MAX_PAYLOAD_LAN] del struct consisteix en el camp de dades de la PDU i el camp de FCS afegit al final. El fet que el camp de FCS no tingui una posició preestablerta fa que no el puguem definir dins del tipus lanpdu_t.

Aquesta definició de tipus permet que en el moment que es tingui una variable de tipus lanpdu_t es pugui reconvertir (fent un canvi de tipus) a un *missatge_eth_t* molt fàcilment. Pel mateix motiu, quan es tingui un *missatge_eth_t* també es pot fer referència a ell tractant-lo com una variable lanpdu_t.

5 Implementació mòdul Lan

5.1 API del mòdul

El mòdul Lan és on s'ha d'implementar el protocol de xarxa local que s'ha definit en aquesta pràctica. Aquest mòdul farà servir els mòduls Ether, Error_Morse i Timer. Aquest mòdul ha d'oferir funcions (servei) per permetre una comunicació no fiable entre diferents nodes de la xarxa local, per tant el fitxer de capçalera podria ser el següent

```
#ifndef LAN_H
#define LAN_H
#include <inttypes.h>
#include <stdbool.h>
#include <pbn.h>

#define MAX_MSS_LAN (MAX_ME_ETH - 4)
//4 Bytes are overload of Lan layer pdu

typedef morse_c_t message_lan_t[MAX_MSS_LAN];

void lan_init(morse_c_t no); //no is origin node ID

bool lan_can_put(void);
void lan_block_put(const message_lan_t m, morse_c_t nd);

//Callback when a message is received
typedef void (*lc_messrx_t)(void);
morse_c_t lan_block_get (message_lan_t m);
void on_lan_received (lc_messrx_t cm);

#endif
```

Com es pot observar, aquest mòdul ofereix pràcticament les mateixes funcions que ofereix el mòdul Ether, afegint la identificació pròpia i la identificació del destinatari dels missatges que es volen transmetre.

void lan_init(morse_c_t no) inicialitza el mòdul i per tant el protocol. Té com paràmetre *no* que és l'adreça origen del propi node.

bool lan_can_put(void) informa si lan_block_put() es pot cridar o no.

void lan_block_put(const message_lan_t m,morse_c_t nd) té dos paràmetres. El primer és el missatge que es vol transmetre. El segon és l'adreça del node a qui va dirigit.

morse_c_t lan_block_get(message_lan_t m) té com a paràmetre un *message_lan_t* on es recollirà el missatge. També retorna l'adreça del node que ha enviat el missatge.

on_lan_received(lc_messrx_t cm) Aquesta funció serveix per instal·lar la funció de callback que es passa com a paràmetre. Aquesta funció de callback serà cridada quan la capa Lan tingui un missatge disponible per lliurar. Cal fer notar que a la definició d'aquesta funció de callback haurà la crida a lan_block_get().

5.2 Implementació de la capa Lan

5.2.1 La transmissió

Des del punt de vista de transmissió, quan la capa d'aplicació lliura a la capa (mòdul) Lan un *message_lan_t*, el mòdul Lan construeix una *lanpdu_t* que lliurarà a la capa Ether (amb la crida a *ether_block_put()*) tipificada com un *missatge_eth_t*.

En aquest protocol es fan successius intents de transmissió mentre es troba el canal ocupat, sense límit en el nombre d'intents. Si la comunicació acaba sent inviable, serà l'usuari qui resolgui la situació avortant l'execució.

Per implementar la part corresponent a la transmissió de la capa LAN es fa servir un autòmat. Anomenarem a aquest autòmat *tx_autòmat_lan*. Implementarem aquest autòmat com un funció que no retorna res però que té com a paràmetre un esdeveniment. Aquest autòmat farà les accions oportunes en funció dels esdeveniments que passen i l'estat del sistema. Per tant, el primer pas és identificar quin són els possibles estats del sistema i quins són els possibles esdeveniments que poden passar.

Comencem per identificar els esdeveniments on estigui implicada la part transmissora de la capa LAN:

1. Que la capa superior sol·liciti la transmissió d'un missatge per part de la LAN. Aquest esdeveniment està íntimament relacionat amb la crida de la funció lan_block_put(). Això implica que la implementació de la funció lan_block_put() haurà de generar aquests esdeveniment. La manera de generar-lo és cridar a l'autòmat de transmissió de la LAN passant com a paràmetre aquest esdeveniment. A aquest esdeveniment l'anomenem e_tx.
2. Que hagi passat el temps aleatori després d'un intent de transmissió amb el canal Ether ocupat. Anomenem a aquest esdeveniment e_tout.

Per altra banda, s'ha de identificar els possibles estats del sistema. En aquest cas, el mateix *tx_autòmat_lan* es pot trobar en 2 possibles estats:

1. Estat de repòs, corresponent a l'estat inicial, on està disponible per poder transmetre un missatge. Anomenat s_idle.
2. Estat de pendent d'enviar algun missatge degut a que no s'havia pogut enviar anteriorment perquè la capa ether no estava disponible. Anomenat s_trying.

El comportament d'aquest autòmat també variarà en funció de l'estat en que també es trobi una altra part del sistema, la corresponent a l'estat de l'ether. És a dir, l'autòmat actuarà de manera diferent en funció de si el canal de comunicacions està lliure o no. Aquest estat es pot conèixer per la funció ether_busy(). Anomenarem als estats de l'ether s_efree i s_ebysy. Dit això, la conseqüència final és que els possibles estats són la combinació d'aquest altres subestats. Per tant, en total seran 4 possibilitats.

L'evolució de la transició entre els estats vindrà donat pel següent comportament:

- L'estat inicial serà el de repòs
- Si es produeix tant l'esdeveniment e_tx com e_tout i l'estat de l'ether és s_efree es transferirà el missatge a l'ether per a que faci la transmissió i es passarà a l'estat s_idle.
- Si es produeix tant l'esdeveniment e_tx com e_tout i l'estat de l'ether és s_ebusy es crearà un altre esdeveniment temporitzat aleatori per fer un nou intent de transmissió i es passarà a l'estat s_trying.

Si el node està en s_trying no permet que la capa superior li sol·liciti un nou enviament.

Per ajudar a la implementació d'aquest autòmat, disposeu dels següents fragments de codi:

```
typedef enum {s_idle, s_trying} s_txlan_t;
typedef enum {s_efree, s_ebusy=10} s_ether_t;
typedef enum {e_tx, e_tout} e_txlan_t;

static void tx_automat_tout (void) {
        tx_automat_lan(e_tout);
}

static void tx_automat_lan (e_txlan_t e) {
        s_ether_t s_ether;
        ...

        s_ether = (ether_busy() ? s_ebusy : s_efree);
        switch (s_txlan + s_ether) {
        case s_idle + s_efree: {
                ...
        ...
}

void lan_block_put (const message_lan_t m, morse_c_t nd) {
        ...
        //Fer el que calgui
        tx_automat_lan(e_tx);
}
```

5.2.2 La recepció

Des del punt de vista de recepció, quan un missatge viatja pel canal de comunicacions i arriba a un dispositiu, la part receptora del mòdul (capa) Ether recull aquest missatge com un *missatge_eth_t*, aquest mateix missatge el lliura al mòdul Lan que el considera (amb

el canvi de tipus necessari) un *lanpdu_t*. Finalment, si tot és correcte, d'aquest *lanpdu_t* lliurarà el camp de dades a la capa d'aplicació tipificada com un *message_lan_t*.

El lliurament dels missatges es gestionarà via callback, per tant no s'haurà d'implementar cap mena de mecanisme d'enquesta o de comprovació per saber si la capa Ether ha rebut algun missatge, ja que serà el mateix mòdul ether qui executarà la funció de callback quan l'Ether hagi rebut un missatge. La funció que se li passa a la Ether serà una funció que haurà de comprovar si el missatge és per a ell i si no té cap error. En cas contrari descartà el missatge. En el cas que es compleixin aquestes condicions aquesta mateixa funció haurà de lliurar el missatge rebut a la capa superior també via una nova funció de callback definida a la capa d'aplicació. Aquesta última funció de callback definida a la capa d'aplicació cridarà a *lan_block_get()* per recollir el missatge dirigit a la capa d'aplicació i serà instal·lada amb la crida a *on_lan_received()*.

Aquesta "màgia" s'aconsegueix pel mecanisme de considerar algunes funcions com a paràmetres d'altres funcions. Els fragments de codi que hi ha a continuació esquematitzen aquesta "màgia"

```
static lc_messrx_t deliver_mss = NULL;

static void proc_mess_eth(void) {
        ether_block_get(...);
        //Si és per a mi i no té errors {
                if (deliver_mss) deliver_mss();
        }
}

void on_lan_received (lc_messrx_t f) {
        deliver_mss = f;
}

...
        on_message_received(proc_mess_eth);
```

6 Capa d'aplicació

El protocol de xarxa local d'aquesta pràctica oferirà servei a les capes superiors. En aquest cas, la capa superior serà directament la capa d'aplicació. Per tant, per poder fer una aplicació de comunicacions s'ha de definir que ha de fer la capa l'aplicació de la pila de comunicacions que estem definint.

L'aplicació consistirà en un xat entre tots els nodes de la xarxa local. Com interfície d'usuari, es farà servir la pantalla i el teclat del vostre ordinador personal. Així, la feina del vostre ordinador serà la d'establir un pont entre teclat/pantalla i la interfície sèrie

de l'Arduino. Aquesta funcionalitat del vostre ordinador la durà a terme un programa terminal com el picocom. D'aquesta manera no cal desenvolupar cap aplicació per a l'ordinador personal.

6.1 Requeriments

6.1.1 Recepció canal morse - transmissió canal sèrie (pantalla ordinador)

Els requeriments de recepció morse de l'aplicació a la banda del AVR són els següents:

- Qualsevol missatge rebut pel canal de comunicacions morse, que sigui lliurat a la capa d'aplicació, s'enviarà pel port sèrie, permeten així la seva visualització. Per millorar la presentació cap a l'usuari a través de l'aplicació de terminal de l'ordinador personal, cada missatge estarà acabat en un retorn de carro i avançament de línia fent que cada missatge ocupi 1 línia nova de pantalla.
- En aquestes línies s'ha d'indicar qui és l'origen i el destí del missatge de manera que els primers caràcters de la línia seran "No->Nd:" on No és el caràcter morse corresponent a l'adreça del node origen i Nd és el caràcter morse corresponent a l'adreça del node destí.
- La resta de la línia serà el contingut del missatge.

6.1.2 Transmissió del canal morse - recepció canal sèrie (teclat ordinador)

Els requeriments en transmissió morse són:

- L'usuari teclejarà el missatge que vol transmetre en el terminal del seu ordinador personal. Aquest missatge serà rebut pel port sèrie de l'AVR. El funcionament típic d'un terminal d'ordinador implica que cada caràcter teclejat és automàticament enviat pel port sèrie. Això vol dir que l'AVR anirà rebent el missatge caràcter a caràcter i no de cop en forma de bloc.
- Per tal d'indicar a quin node va dirigit el missatge, el format del missatge que escriu l'usuari des del terminal consisteix en els següents caràcters:
  - caràcter corresponent a l'adreça del node destí
  - ":"
  - caràcters del missatge pròpiament dit
  - indicació de final de missatge fet amb el caràcter de retorn de carro
- Opcionalment, per tal d'avortar l'escriptura de missatge actual i tornar a començar l'escriptura d'un nou missatge, es contempla l'enviament de la lletra "r" en minúscula. Això provoca que l'AVR descarti el missatge actual i comenci de nou a rebre un nou missatge. Com a resposta cap a l'usuari, l'AVR enviarà el missatge "RESET" pel canal sèrie.

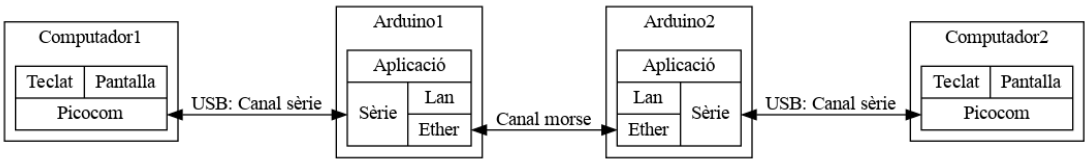
6.2 Implementació de la capa d'aplicació

Des del punt de vista de la transmissió morse, un autòmat ha d'anar processant els caràcters que es van rebent pel port serie fins trobar un retorn de carro o "r". Aquest processament es limita a considerar el primer caràcter com l'adreça de destí, el segon caràcter un ":" i a partir d'aquí hi ha el missatge que es vol transmetre. Quan es rep retorn de carro vol dir que hi ha un missatge per ser enviat. Si el missatge té el format adequat se sol·licita a la capa Lan la seva transmissió. Si la capa Lan no permet la transmissió, el programa queda esperant i bloquejat fins que la capa Lan permeti la transmissió.

Des del punt de vista de la recepció morse, el missatge rebut a través del canal morse que ha lliurat la capa Lan a l'aplicació s'ha de reenviar, aprofitant la implementació a través del callback, cap el port sèrie. Així, el missatge serà presentat en el terminal de l'ordinador de l'usuari. Aquest missatge s'ha de formatar segons les especificacions del punt 6.1.1, afegint l'adreça d'origen i de destí.

7 Pila de comunicació

El sistema de comunicació que s'està definint a l'AVR està estructurat per capes. Aquestes capes de comunicació s'implementen en C a través de mòduls. El fet de considerar capes de comunicació als mòduls on s'implementen protocols de comunicació es deu a que cada mòdul depèn d'altres mòduls de forma jeràrquica i sense barrejar-se entre ells.



L'esquema anterior indica que el mòdul Aplicació depèn del mòdul Lan, però no té cap dependència amb el mòdul Ether. Per altra banda, el mòdul Lan depèn del mòdul Ether. D'aquesta manera queda definida una estructuració per capes on cada capa depèn del que ofereix la seva capa inferior, però no altres que queden més per sota. També s'observa que la capa d'Aplicació també fa servir la capa Sèrie (formada per *serial* i *blck_serial*).

La capa d'aplicació serà implementada en el mòdul principal, que és on està definida la funció main(). La resta de capes seran implementades en mòduls separats, utilitzant màquines d'estat i sense definir funcions bloquejants.

L'esquema mostra tots els dispositius que intervenen en el muntatge, incloent els canals de comunicació.

8 Treball pràctic

En aquesta pràctica, es considera el CRC com l'algoritme utilitzat per a la detecció d'errors.

8.1 Sessió 1

1. Dibuixa el graf de la màquina d'estat corresponent a la recepció sèrie de la capa d'aplicació
2. Defineix les possibles funcions i variables privades de la capa d'aplicació o mòdul principal.
3. Dissenya la capa d'aplicació suposant que existeix el mòdul Lan.
4. Dissenya un mòdul Lan simplificat que serveixi de curtcircuit entre la capa d'aplicació i la capa Ether per comprovar el correcte funcionament de la capa d'aplicació. En aquest cas, qualsevol missatge que es rebi per la capa Ether es lliurarà directament a l'aplicació. Els identificadors de node origen i destí deixen de tenir sentit, per tant tindran valors arbitraris.
5. Comprova el correcte funcionament de la capa d'aplicació amb el mòdul Lan simplificat i el canal Ether ideal amb cable directe sense infraroig. Calen 2 Arduinos per aquesta comprovació. Si la implementació és correcta, aquests 2 Arduinos poden ser de diferents grups.
6. Repeteix la comprovació de l'apartat anterior incorporant el canal Ether real (amb infraroig).

8.2 Sessió 2

1. Dibuixa el graf de la màquina d'estat corresponent a la transmissió de la capa Lan
2. Defineix les possibles funcions i variables privades del mòdul Lan de la part transmissora per estructurar millor el disseny de manera que quedi el més simple possible.
3. Dissenya les funcions de transmissió del mòdul Lan i incorpora-les al mòdul Lan simplificat de la sessió anterior.
4. Comprova el correcte funcionament de la part transmissora del mòdul Lan amb 2 Arduinos i el canal Ether ideal. La part receptora mostrada per l'aplicació correspondrà directament als missatges de la capa Ether (ja que es manté el curtcircuit amb Ether en recepció). En conclusió, els missatges rebuts correspondran a la lanpdu_t creada per la part transmissora de la Lan.
5. Repeteix la comprovació a de l'apartat anterior amb el canal Ether real (amb infraroig).

8.3 Sessió 3

1. Dissenya les variables i funcions privades així com les funcions públiques de recepció del mòdul Lan i incorpora-les al mòdul Lan.
2. Comprova el correcte funcionament del mòdul Lan amb 2 Arduinos i el canal Ether ideal. Si la implementació és correcta, aquests 2 Arduinos poden ser de qualsevol grup.
3. Repeteix la comprovació de l'apartat anterior amb el canal Ether real. Observeu que ara la comunicació ignorarà els missatges amb errors i que poden haver més de 2 Arduinos compartint el mateix canal de comunicació.

Referències

[avr-libc] `http://www.nongnu.org/avr-libc/user-manual/modules.html`