Xarxes de Comunicació

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

Francisco del Águila López

Setembre 2017

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 creant d'aquesta manera 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 Ether

La natura de la transmissió per àudio en Morse fa que el canal àudio 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 i disposar, per exemple, de la funció *ether_can_put()* que determina si el pot transmetre o no un missatge.

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

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

typedef uint8_t *block_morse;
typedef void (*ether_callback_t)(void);

void ether_init(void);

bool ether_can_put(void);
void ether_block_put(const block_morse b);

bool ether_can_get(void);
void ether_block_get(block_morse b);
void on_message_received(ether_callback_t m);
void on_finish_transmission(ether_callback_t f);

#endif
```

typedef uint8_t *block_morse És una definició de tipus que contempla el bloc de caràcters codificats en ASCII que s'enviaran pel canal Morse. La capa d'enllaç serà la responsable de crear una variable en forma de taula de *block_morse* amb el contingut dels caràcters que es voldrà enviar o rebre. En el moment de definir la mida d'aquesta taula, es considerarà el valor màxim de la cua existent al mòdul *Ether* per poder enviar els caràcters. En aquest cas, considerarem un valor de 32 caràcters (incloent els bytes afegits de control). El motiu de la nova definició de tipus és perquè els valors vàlids que pot contenir aquest *block_morse* són: els caràcters corresponents als números 0..9 i els caràcters corresponents a les lletres majúscules A..Z

void ether_init(void) Serveix per inicialitzar el canal físic.

bool ether_can_put(void) Es farà servir per comprovar si és possible la transmissió d'un *block_morse*. En el cas que s'està treballant (transmissió punt a punt), vindrà determinada essencialment per si s'està rebent algun *block_morse* enviat per un altre node.

void ether_block_put(const block_morse b) Aquesta funció es fa servir per transmetre un *block_morse*. Se li passa com a paràmetre l'apuntador a taula de *uint8_t* on són els caràcters que es vol transmetre. S'ha de considerar que l'indicador de fins a on està plena aquesta taula ve determinat pel byte *nul* ('\0') que farà de sentinella.

bool ether_can_get(void) És una funció que indicarà quan està disponible tot un

block_ morse. Aquesta funció es fa certa quan ha arribat l'últim byte del block. Funciona diferent a la original, que indicava disponibilitat de lectura a partir del moment que es rebia el primer caràcter.

void ether_block_get(block_morse b) Aquesta funció ofereix la possibilitat de llegir el contingut del *block_ morse* rebut. El protocol d'enllaç ha de reservar una taula de tipus *block_ morse* i aquesta funció la omplirà amb el contingut del *block_ morse* rebut. El paràmetre que se li passa és precisament l'apuntador a aquesta taula. 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. Si s'instal·la el callback la funció ether_can_get() perd la seva utilitat. A més, utilitzant 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. De la mateixa manera, només hi ha com a paràmetre l'apuntador a la taula de bytes, però es podria afegir també la mida fins on està plena aquesta taula. En el nostre cas no es fa necessària aquesta característica ja que el sentinella ja fa aquesta funció. En canvi, aquest paràmetre de mida d'ocupació de taula, es fa imprescindible en el cas que la transmissió fos binaria i el caràcter *nul* es podria confondre amb una dada més.

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. Disposa d'un *buffer* de 32 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.

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 10 ms. It is based on callbacks. That is, functions which
 * are called after a specific (temporal) event ocurred.
 */
#define TIMER_MS(ms) (ms/10)
#define TIMER_ERR    -1

typedef void (*timer_callback_t)(void);
typedef int8_t timer_handler_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);

#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 10ms dóna un temps màxims de 655 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.

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. Entrega aquestes dades a la capa superior.

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 event temporal (amb el mòdul timer) per tornar a fer l'intent de transmissió passat aquest temps aleatori.
- Repeteix aquest procés 2 vegades més com a màxim fins que aconsegueix la transmissió o bé descarta aquesta transmissió i encén el led indicant l'error.

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. Es reserva un caràcter morse pel seu valor. S'ha de gestionar adequadament aquestes adreces per evitar la duplicitat d'adreces a la mateixa xarxa local.

Adreça de destí És el camp corresponent a identificar el node a qui se li envia la informació. Es reserva un caràcter morse pel seu valor.

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.

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

```
typedef struct {
    uint8_t origen;
    uint8_t desti;
    uint8_t payload[30];
} lanpdu_t;
```

S'ha de tenir en compte que el camp payload[30] 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 al tipus lanpdu_t.

Aquesta definició de tipus permet que en el moment que es tingui una variable de tipus lanpdu_t es pugui reconvertir a un block_morse molt fàcilment. Pel mateix motiu, quan es tingui un block_morse 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>

typedef void (*lan_callback_t)(void);
typedef uint8_t * missatge_lan_t

void lan_init(uint8_t no);

bool lan_can_put(void);
void lan_block_put(const missatge_lan_t m, uint8_t nd);

//bool lan_can_get(void);
uint8_t lan_block_get(missatge_lan_t m);
void on_lan_received(lan_callback_t l);

#endif
```

Com es pot observar, aquest mòdul ofereix les mateixes funcions que ofereix el mòdul Ether. Però en aquest cas permet identificar amb quin node es vol realitzar la comunicació.

Quan es vol transmetre un bloc de dades utilitzant el mòdul Ether, es fa servir la funció ether_block_put() que requereix com a paràmetre un apuntador a una taula de tipus *block_morse*. Per aquest motiu s'ha de reservar un espai de memòria on estigui aquesta taula. Aquesta reserva de memòria es podria fer dinàmicament (*malloc()*)o estàticament (variable global). Per simplicitat es farà estàticament, per tant en el mòdul s'ha de definir una variable global i privada del mòdul que serà la taula de *block_morse* que s'utilitzarà per crear la trama de transmissió. El mòdul Ether tracta aquesta taula com un *block_morse,* però la mateixa taula serà tractada pel mòdul Lan com un *lan_pdu_t.*

De la mateixa manera, quan es vol rebre un bloc de dades utilitzant el mòdul Ether, es fa servir la funció ether_block_get(), per tant també s'ha de definir una variable global i privada que serà la taula de *block_morse* per a la recepció. Des del punt de vista del mòdul Lan, aquesta mateixa taula també serà tractada com un *lan_pdu_t.*

En aquest protocol es fan successius intents de transmissió mentre es troba el canal ocupat. El número màxim de intents serà de 3.

void lan_init(uint8_t no) inicialitza el mòdul i per tant el protocol. Té com paràmetre *no* que és l'adreça origen del propi node.

void lan_block_put(const missatge_lan_t m,uint8_t nd) té dos paràmetres. El primer és el missatge que es vol transmetre en forma de taula *missatge_lan_t.* El

segon és l'adreça del node a qui va dirigit.

uint8_t lan_block_get(missatge_lan_t m) té com a paràmetre la taula *missatge_lan_t* on es recollirà el missatge. També retorna l'adreça del node que ha enviat el missatge. Aquesta funció està molt relacionada amb lan_can_get() ja que aquesta última indica quan hi ha disponible un missatge per llegir i per tant quan es pot cridar a lan_block_get().

Una alternativa a implementar la funció *lan_can_get()* és instal·lar una funció de callback *on_lan_received()* que serà cridada just quan hi hagi dades disponibles. Per aquest motiu la funció *lan_can_get()* està comentada en el fitxer de capçalera, ja que la implementació recomanable i elegant és fer servir les funcions de callback.

5.2 Implementació de la capa lan

5.2.1 Punt de vista de transmissió

Cada node pot estar en dos possibles estats: "pendent_enviar" o "esperant". En l'estat "esperant" simplement espera que la capa superior li demani l'enviament d'algun missatge. Mentrestant està pendent de la rebuda de possibles missatges.

Si la capa superior li demana l'enviament d'un missatge, el node passa a l'estat "pendent_enviar" fins que aconsegueix l'enviament. Si ho aconsegueix just en el mateix moment, torna a l'estat "esperant" de nou. Però si això no és possible, ha de tornar-ho a intentar 2 vegades més programant events temporitzats. Quan ho aconsegueix torna a l'estat "esperant". En cas que no ho aconsegueixi, després dels intents disponibles, indica la condició d'error encenent el led. L'estat que queda quan s'ha produït aquest error es deixa a la decisió de l'estudiant.

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

5.2.2 Punt de vista de recepció

En aquest cas, quan un node rep un missatge i compleix amb les condicions de no tenir errors, si el missatge que rep no és per ell el descarta i no fa res més. Però si el missatge que rep sí que és per ell, activa la crida del callback corresponent o indica que hi ha missatges pendents de ser llegits per poder lliurar-los a la capa superior.

6 Capa d'aplicació

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

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 de 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

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

- Qualsevol missatge rebut pel canal de comunicacions morse, que compleixi la condició de ser lliurat a la capa superior, 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 fent que cada missatge ocupi 1 línia 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 (uint8_t) 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

Els requeriments en transmissió 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, 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 i aprofitant la implementació a través del callback, l'aplicació tan sols ha de reenviar el missatge rebut pel canal morse i lliurat per la capa Lan cap el port sèrie per ser presentat en el terminal de l'ordinador de l'usuari. El missatge lliurat per la capa Lan s'ha de formatar segons les especificacions del punt 6.1.1, afegint l'adreça d'origen i de destí.

Si per contra, es fa una implementació no basada en callback, el programa principal ha d'anar contínuament consultant si hi ha dades disponibles amb lan_can_get() per presentar-les pel port sèrie quan estiguin disponibles.

7 Treball pràctic

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

1. Dibuixa el graf de la màquina d'estat corresponent a la transmissió a la capa d'aplicació
2. Dibuixa el graf de la màquina d'estat corresponent a la transmissió a la capa Lan.
3. Defineix les possibles funcions i variables privades del mòdul Lan per estructurar millor el disseny de manera que quedi el més simple possible.
4. Defineix les possibles funcions i variables privades del programa principal.
5. Dissenya l'aplicació final suposant que existeix el mòdul Lan.
6. Dissenya les funcions de recepció del mòdul Lan. Comprova el correcte funcionament del mòdul Lan amb l'aplicació final en un arduino i amb un programa transmissor de test que generi les trames de manera concreta en un altre arduino.
7. Dissenya les funcions de transmissió del mòdul Lan. Comprova el seu correcte funcionament amb l'aplicació final.

8. Comprova el correcte funcionament combinant la transmissió i la recepció.

Referències

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